UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

LUCAS CARDOZO JANTSCH

PROJETO DE DIPLOMAÇÃO

DESENVOLVIMENTO DE UM CONTROLADOR DE MOTORES BRUSHLESS DC BASEADO EM FPGA PARA USO EM VEÍCULOS ELÉTRICOS

Porto Alegre
2021

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

DESENVOLVIMENTO DE UM CONTROLADOR DE MOTORES BRUSHLESS DC BASEADO EM FPGA PARA USO EM VEÍCULOS ELÉTRICOS

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul, como requisito parcial à obtenção do título de Bacharel em Engenharia Elétrica.

ORIENTADOR: Prof. Dr. Tiago Roberto Balen

Porto Alegre
2021

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

LUCAS CARDOZO JANTSCH

DESENVOLVIMENTO DE UM CONTROLADOR DE MOTORES BRUSHLESS DC BASEADO EM FPGA PARA USO EM VEÍCULOS ELÉTRICOS

Este projeto foi julgado adequado para fazer jus aos créditos da Disciplina “Projeto de Diplomação”, do Departamento de Engenharia Elétrica, e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador:_____
Prof. Dr. Tiago Roberto Balen, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul

Banca Examinadora:

Prof. Dr. Tiago Roberto Balen, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul

Prof. Dr. Ivan Müller, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul

Prof. Dr. Jeferson Vieira Flores, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul

Porto Alegre, Maio de 2021.

DEDICATÓRIA

Dedico este trabalho aos meus pais. Sua grande força foi a mola propulsora que permitiu o meu avanço, mesmo durante os momentos mais difíceis.

AGRADECIMENTOS

A Deus, pela saúde e força para superar as dificuldades.

Aos meus pais, pelo amor, incentivo e apoio incondicional em todos os momentos de minha vida.

Ao professor Dr. Tiago Roberto Balen, pela orientação, assistência e confiança durante a elaboração deste trabalho.

A todos os familiares, que nunca negaram palavras de incentivo ao longo desta jornada acadêmica.

Ao corpo docente desta instituição, pela sabedoria, paciência e esforço na transmissão do conhecimento.

A esta universidade, por proporcionar a estrutura necessária para que pudesse crescer pessoal e academicamente.

A todos os amigos e colegas, que me encorajaram durante a realização do curso de engenharia.

A todos que direta ou indiretamente fizeram parte de minha formação, o meu muito obrigado.

“Creio firmemente em uma lei de compensação. As verdadeiras recompensas são sempre proporcionais ao esforço e aos sacrifícios feitos.”

Nikola Tesla

RESUMO

Neste trabalho é desenvolvido em linguagem VHDL o sistema digital de um controlador de motores *Brushless* DC baseado em FPGA para uso em veículos elétricos. O objetivo principal é desenvolver e implementar em um FPGA um controlador de motores *Brushless* DC. O objetivo secundário é caracterizar parâmetros como tensão, corrente e potência elétrica em um veículo elétrico equipado com o controlador desenvolvido. Foi realizada uma pesquisa na literatura especializada da área de forma a caracterizar o motor *Brushless* DC do ponto de vista construtivo e operacional. De forma conjunta, foram investigadas as arquiteturas existentes para: veículos elétricos e dispositivos FPGA. O desenvolvimento deste projeto foi elaborado através das seguintes etapas: requisitos de projeto, arquitetura do sistema, materiais utilizados, sistema digital desenvolvido e protótipo desenvolvido. Através dos testes realizados, identificou-se que o controlador desenvolvido é capaz de controlar o sentido de rotação e velocidade de um motor *Brushless* DC, obtendo ainda erro nulo em regime permanente. Os experimentos conduzidos com o protótipo de veículo elétrico elaborado permitiram quantificar o consumo energético por distância percorrida, onde observou-se o valor máximo de 13,9 Wh/km para uma velocidade de 10 km/h considerando um conjunto veículo-usuário com aproximadamente 100 kg de massa.

Palavras-chave: Controlador de motores *Brushless*. *Brushless* DC. FPGA. Veículos elétricos.

ABSTRACT

In this work, the digital system of a Brushless DC motor controller based on FPGA for use in electric vehicles is developed in VHDL language. The main objective is to develop and implement a Brushless DC motor controller in an FPGA. The secondary objective is to characterize parameters such as voltage, current and electrical power in an electric vehicle equipped with the developed controller. A search was carried out in the specialized literature of the area in order to characterize the Brushless DC motor from a constructive and operational point of view. Together, the existing architectures for: electric vehicles and FPGA devices were investigated. The development of this project was elaborated through the following steps: design requirements, system architecture, materials used, digital system developed and prototype developed. Through the tests performed, it was identified that the developed controller is capable of controlling the direction of rotation and speed of a Brushless DC motor, still obtaining null error in steady state. The experiments conducted with the elaborated electric vehicle prototype allowed to quantify the energy consumption by distance traveled, where the maximum value of 13.9 Wh/km was observed for a speed of 10 km/h considering a vehicle-user set with approximately 100 kg of mass.

Key-words: Brushless motor controller. Brushless DC. FPGA. Electric Vehicles.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

DC: *Direct Current*

AC: *Alternating Current*

BLDC: *Brushless DC*

FPGA: *Field Programmable Gate Array*

VHDL: *VHSIC Hardware Description Language*

VHSIC: *Very High Spedd Integrated Circuit*

EUA: Estados Unidos da América

DSP: *Digital Signal Processor*

MCU: *Microcontroller Unit*

GEE: Gases do Efeito Estufa

VCI: Veículo à Combustão Interna

HEV: *Hybrid Electric Vehicle*

PEV: *Plug-in Electric Vehicle*

PHEV: *Plug-in Hybrid Electric Vehicle*

E-REV: *Extended-Range Electric Vehicle*

BEV: *Battery Electric Vehicle*

FCEV: *Fuel Cell Electric Vehicle*

PMSC: *Permanent-Magnet Synchronous Motor*

BACM: *Brushless AC Motor*

BDCM: *Brushless DC Motor*

BEMF: *Back Electromotive Force*

ASICs: *Application Specific Integrated Circuits*

LUT: *Look up Table*

RAM: *Random Access memory*

PID: Proporcional-Integral-Derivativo

PI: Proporcional-Integral

PWM: *Pulse Width Modulation*

RPM: Rotações Por Minuto

VGA: *Video Graphics Array*

IR: *Infrared*

LCD: *Liquid Crystal Display*

PLL: *Phase-Locked Loop*

USB: *Universal Serial Bus*

SUMÁRIO

1. INTRODUÇÃO

1.1. CONTEXTUALIZAÇÃO E MOTIVAÇÃO

Nas últimas décadas, a população mundial vem apresentando um crescimento cada vez mais rápido, o que gera um impacto cada vez maior sobre o meio em que vivemos, tanto sob o aspecto de ocupação territorial como sob a utilização dos recursos naturais.

A preocupação sobre os impactos da atividade do homem no meio ambiente é cada vez maior, devido à limitação dos recursos naturais disponíveis e os impactos causados pela sua extração.

Esses fatores reforçam a necessidade de novas pesquisas científicas, a fim de gerar tecnologias capazes de satisfazer as exigências energéticas, atuais e futuras, minimizando o impacto ambiental no uso dos recursos disponíveis.

Uma grande parcela da energia utilizada atualmente vem de fontes não renováveis como o petróleo, que é líder principalmente no segmento de transportes. Conforme os dados da Agência de Administração de Informações sobre Energia dos EUA, 36,5% da energia consumida pelo país em 2018 veio do petróleo, e desta parcela aproximadamente 69% foi utilizado para fins de transporte. (Energy Information Administration, 2019).

O uso de petróleo como principal fonte de energia em veículos apresenta um problema de sustentabilidade, pois se trata de um recurso não-renovável, que pode se tornar escasso no decorrer do tempo. Segundo estimativas do centro de pesquisas em energia do Reino unido, que analisou dados de mais de 500 estudos na área de esgotamento de reservas de óleo juntamente com bancos de dados da indústria petrolífera, existe uma grande probabilidade de se observar um pico na produção de petróleo mundial entre 2020 e 2030, seguido por um rápido declínio anual na taxa de produção de óleo, causado pelo esgotamento progressivo das reservas. (SORRELL et al., 2010).

Uma solução para o problema de suprimento energético do setor de transportes pode ser obtida ao encontrar formas alternativas de fornecimento de energia. Dessa forma, novas tecnologias têm sido estudadas a fim de satisfazer os principais aspectos competitivos: autonomia e preço. (KHALIGH; LI, 2010).

Uma dessas tecnologias já está sendo utilizada em alguns dos mais novos veículos produzidos, que é a propulsão elétrica com uso de baterias. A vantagem desta tecnologia é sua eficiência, que segundo a Fonte Governamental de Informações para Economia de Combustível dos EUA, ultrapassa os 77%, muito maior que a dos tradicionais veículos à combustão, que apresentam entre 12 e 30%. (U.S. Department of Energy, 2011).

Esse alto nível de eficiência é obtido principalmente pela utilização de um tipo de motor elétrico específico, conhecido como *Brushless* DC (BLDC). A grande vantagem deste tipo de motor é a ausência de escovas, que são contatos deslizantes destinados a conduzir a energia elétrica à parte móvel dos motores DC convencionais. A ausência destes contatos representa uma grande vantagem, pois resulta em menores perdas elétricas e uma menor necessidade de manutenção. (GIERAS; WING, 2002).

Essas vantagens possuem um preço, que é a maior complexidade no acionamento deste motor. Para tal função podem ser utilizados dispositivos semicondutores (transistores bipolares de porta isolada, transistores de efeito de campo ou tiristores em comutação forçada). (JULIANI, 2009). O acionamento destes componentes deve ser realizado de acordo com a posição angular do rotor, que pode ser obtida por Sensores do tipo Hall ou pela medição da força contraeletromotriz produzida nos enrolamentos durante a movimentação do rotor. (GONELLA, 2006).

Para controlar o acionamento eletrônico do motor, diversos dispositivos podem ser utilizados, como por exemplo DSP, MCU ou FPGA. (THOMPSON, 2009).

Dentre estes, um se destaca, o FPGA (*Field Programmable Gate Array*). A grande vantagem deste dispositivo é sua flexibilidade de operação, pois ao contrário das outras tecnologias que possuem hardwares genéricos fixos, um FPGA pode ser customizado para cada tipo de aplicação, de forma a se adequar aos requisitos de performance, segurança e confiabilidade do sistema, utilizando-se de um hardware totalmente reconfigurável internamente.

Outra grande vantagem deste tipo de dispositivo está presente ao nível de integração do sistema, pois o processador pode ser embarcado junto com a lógica de acionamento do motor no mesmo dispositivo FPGA. (ALTERA CORPORATION, 2008).

Pelas vantagens apresentadas, a utilização de um dispositivo do tipo FPGA para controle de motores *Brushless* DC pode representar um ganho de performance e eficiência para os veículos elétricos de forma a torná-los ainda mais atrativos para a sociedade.

1.2. JUSTIFICATIVA E PROBLEMA

A realização deste trabalho se justifica pela crescente preocupação com os impactos ambientais causados pelos veículos movidos à combustão, assim como a busca por veículos mais eficientes e menos agressivos ao meio-ambiente.

Esses fatores têm impulsionado nos últimos anos o desenvolvimento do mercado de veículos elétricos, muito mais eficientes e menos nocivos à natureza.

Desta forma, este trabalho estuda o uso de dispositivos reconfiguráveis do tipo FPGA como uma alternativa para o controle de motores *Brushless* DC em veículos elétricos.

Para que um dispositivo do tipo FPGA possa operar, é necessário que seja desenvolvido um sistema digital que possa ser sintetizado e gravado no dispositivo. Esse sistema digital é o responsável por processar os sinais de entrada e gerar os sinais de saída do dispositivo, definindo assim seu comportamento.

Desta forma, o problema abordado neste trabalho é desenvolver um sistema digital que seja capaz de controlar motores *Brushless* DC através do uso de dispositivos FPGA para aplicação em veículos elétricos.

1.3. OBJETIVOS

O objetivo principal deste trabalho é desenvolver o sistema digital de um controlador de motores *Brushless* DC em linguagem VHDL para aplicação em veículos elétricos movidos à bateria. De forma complementar, o sistema desenvolvido deve ser passível de implementação em um dispositivo FPGA.

O objetivo secundário deste trabalho é caracterizar parâmetros elétricos tais como tensão, corrente e potência elétrica em um veículo experimental equipado com o controlador desenvolvido. Tal caracterização é realizada de duas formas distintas: com e sem carga aplicada ao veículo.

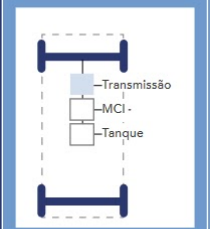
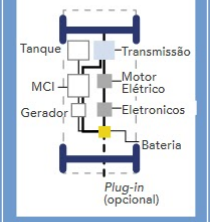
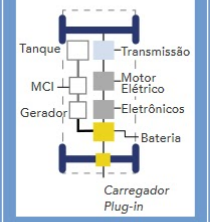
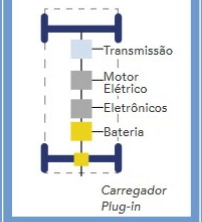
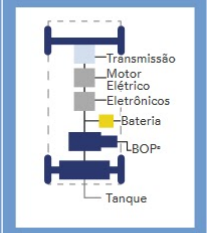
2. REVISÃO BIBLIOGRÁFICA

2.1. ARQUITETURAS DE VEÍCULOS ELÉTRICOS

Com o avanço da tecnologia aplicada em veículos, diversas arquiteturas surgiram com o intuito de melhorar cada vez mais a eficiência dos automóveis e assim, contribuir para a redução das emissões de GEE - gases do efeito estufa.

Na Tabela 1 são apresentadas as principais arquiteturas existentes, onde podem ser observadas suas diferentes características.

Tabela 1: Diferentes arquiteturas de veículos elétricos

	VCI é a fonte primária de propulsão		**Motor elétrico é a fonte primária de propulsão**		
Tipo de veículo	Veículo à combustão interna (VCI) 	Veículo elétrico (*plug in*) híbrido (P)HEV 	Veículo elétrico com autonomia estendida (E-REV ou REX) 	Veículo elétrico à bateria (BEV) 	Veículo elétrico à célula de combustível (FCEV) 
Tipo de motor	Motor à combustão interna	Motores à combustão interna e elétrico dispostos **em paralelo**; sistema *plug in* opcional. Motor à combustão é o principal para mover o veículo, com auxílio de um pequeno motor elétrico	Motores à combustão interna e elétrico dispostos **em série**. Motor elétrico é o principal para mover o veículo, com o motor à combustão interna gerando eletricidade para o elétrico.	100% elétrico	Sistema de célula de combustível e motor elétrico, que propulsiona o veículo, dispostos **em série**.

Fonte: FGV Energia (2017).

O foco deste trabalho são os veículos movidos exclusivamente à bateria, também chamados de BEVs (da sigla em inglês para *Battery Electric Vehicles*).

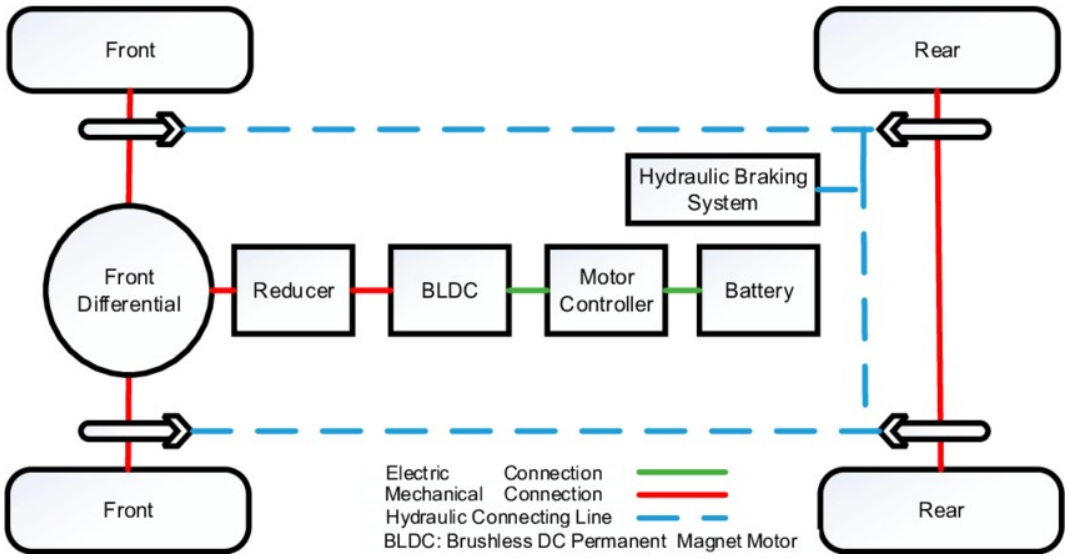
Nessa classe de veículos, a principal fonte de energia é a eletricidade, que normalmente vem de fontes externas, como a rede elétrica. Esses veículos utilizam

uma bateria interna de alta capacidade para armazenar a energia destinada a alimentar o motor elétrico e realizar a propulsão do veículo.

Por utilizar exclusivamente eletricidade para a propulsão, são considerados veículos *all-electric.* Todos os veículos desta classe ainda recebem a denominação de *Plug-in Electric Vehicles* (PEV), visto que a energia utilizada por eles provém de uma fonte externa - o termo *plug-in* vêm do inglês e em uma tradução literal significa "ligado na tomada". (FGV Energia, 2017).

A Figura 1 apresenta a arquitetura de um veículo BEV, onde seus principais componentes podem ser identificados.

Figura 1: Arquitetura de um veículo elétrico movido à bateria



Fonte: XIAO et al. (2017).

Conforme a Figura 1, nesses veículos a conversão de energia elétrica em trabalho mecânico é realizada por um motor do tipo BLDC. Esse motor necessita de um controlador eletrônico através do qual é realizada a conexão da bateria ao motor. A função deste controlador é realizar o acionamento ordenado das bobinas do motor BLDC conforme os parâmetros da aplicação.

2.2. MOTORES *BRUSHLESS*

São motores caracterizados principalmente pela ausência de contatos deslizantes interligando estator e rotor, as chamadas ‘escovas’, (o nome *Brushless* vem do inglês, que significa, ‘sem escovas’). Nestes motores, a excitação magnética é provida por ímãs permanentes instalados no rotor.

Os motores *Brushless* estão contidos na classe dos motores síncronos de ímãs permanentes, denominada PMSC (*Permanent-Magnet Synchronous Motor*). Eles podem ser classificados em duas categorias, são elas: BACM (*Brushless* AC *Motor*) e BDCM (*Brushless* DC *Motor*).

Do ponto de vista construtivo estes motores compartilham as mesmas características, que são: enrolamento de armadura polifásico (geralmente composto por três fases), estator com ranhuras para alojar o enrolamento de armadura e rotor com excitação baseada em ímãs permanentes. A grande diferença entre esses motores está localizada no formato da onda elétrica que é aplicada em cada um. (MONTEIRO, 2002).

Para o BACM é utilizada tensão senoidal polifásica (geralmente trifásica) de forma a operar sob o princípio dos campos girantes, não sendo necessário controlar a posição do rotor para seu acionamento. (GIERAS; WING, 2002).

No BDCM é necessário existir uma realimentação da posição angular do rotor de forma a controlar a corrente nas fases do motor, para permitir o correto sincronismo entre rotor e armadura. (GIERAS; WING, 2002).

Como o foco de estudo deste trabalho é o controle de motores *Brushless* DC, somente estes serão abordados nas próximas seções.

2.2.1. Princípio de operação

De acordo com MILLER (1989), o motor *Brushless* DC de ímãs permanentes possui características de funcionamento semelhantes às máquinas de corrente contínua convencionais, onde essencialmente a corrente do motor percorre as espiras do enrolamento de armadura, produzindo assim um fluxo magnético que interage com o fluxo magnético gerado pelos ímãs permanentes do motor, gerando torque entre o conjunto espira-ímã de forma a estes se alinharem e realizarem a rotação do rotor.

Embora o princípio de operação seja o mesmo entre o motor DC convencional e o motor *Brushless* DC, o mesmo não pode ser dito quanto ao seu acionamento, pois em um motor *Brushless* DC é necessário que a corrente nos seus enrolamentos seja comutada por interruptores eletrônicos de forma a excitar corretamente os enrolamentos de armadura. (JULIANI, 2007).

Outro fator importante no acionamento de motores *Brushless* DC é o sincronismo entre rotor e estator. Deve ser realizado o monitoramento da posição angular do rotor de forma a servir de referência para o correto acionamento dos interruptores eletrônicos. Tal monitoramento pode ser realizado através do método da força contraeletromotriz (em inglês, BEMF - Back ElectroMotive Force), onde é realizada uma medição de tensão nas fases do motor para determinar o correto momento do acionamento, ou ainda pelo método baseado em Sensores do tipo Hall, que são sensores magnéticos instalados entre as ranhuras do estator de forma a detectar a passagem dos polos do rotor. (GONELLA, 2006).

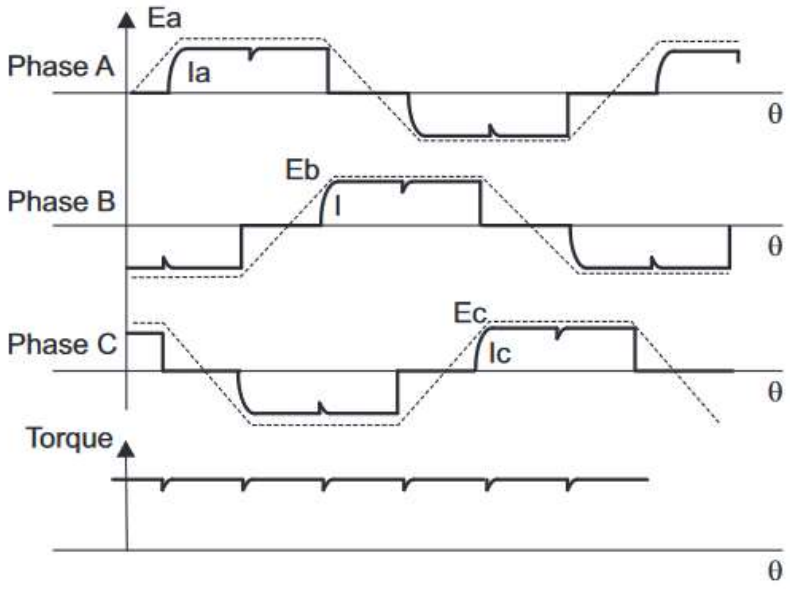
2.2.2. Excitação trapezoidal

Uma das formas mais simples de se acionar um motor *Brushless* DC é utilizar uma forma de onda capaz de energizar de forma cíclica um par de enrolamentos por vez. Uma forma de excitação que atende a esses requisitos se chama comutação trapezoidal 120º, exibida na Figura 2. Ela é assim denominada, pois cada enrolamento acionado permanece nessa condição por 120º elétricos consecutivos.

A principal vantagem desse método de excitação é a possibilidade da obtenção de torque constante. Outra vantagem apresentada por este método de acionamento é a presença de um enrolamento desenergizado a cada instante no motor, o que permite a identificação da posição do rotor através da tensão elétrica induzida neste enrolamento (método da Força ContraEletroMotriz).

Apesar de teoricamente possível, na prática o torque apresentado pelo motor possui certo *ripple* associado, visto que não é possível estabilizar instantaneamente a corrente elétrica nos enrolamentos no momento da comutação do motor. Dessa forma, a cada 60º elétricos observa-se uma ligeira variação no torque disponível. (AKIN; BHARDWAJ, 2011).

Figura 2: Formato da excitação trapezoidal 120º

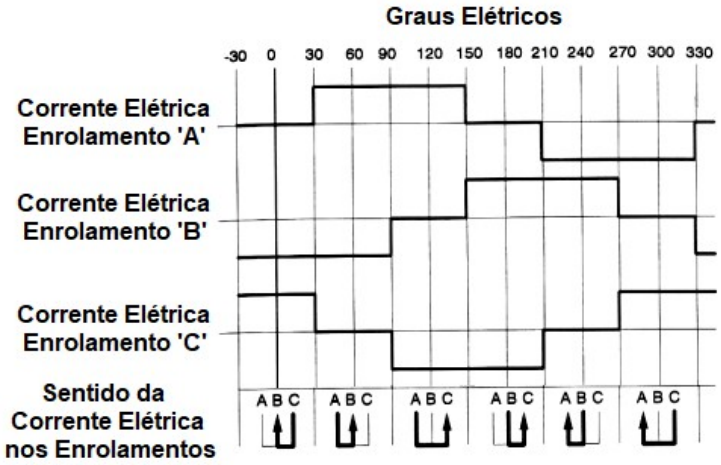


Fonte: AKIN; BHARDWAJ (2011).

2.2.3. Sequência de comutação

A Figura 3 exibe um gráfico contendo a sequência de comutação dos enrolamentos do motor em graus elétricos e a forma de onda observada sobre estes na excitação trapezoidal 120º. Para o acionamento do motor em sentido oposto, basta executar a sequência também em sentido oposto.

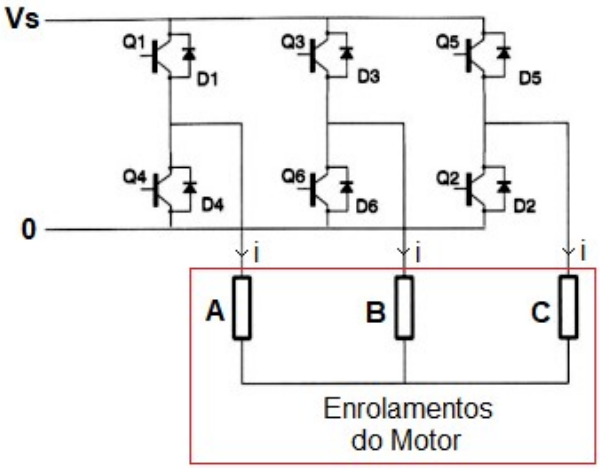
Figura 3: Sequência da comutação trapezoidal 120º



Fonte: Adaptado de: HENDERSHOT; MILLER (1994).

A Figura 4 exibe a topologia típica utilizada para o acionamento dos enrolamentos do motor, uma Ponte Inversora Trifásica com transistores.

Figura 4: Topologia do circuito de acionamento



Fonte: Adaptado de: HENDERSHOT; MILLER (1994).

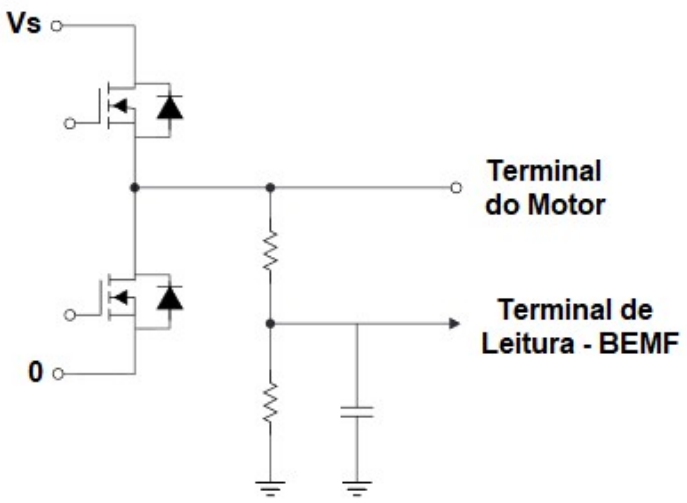
2.2.4. Detecção da posição do rotor - BEMF

Nessa técnica é realizada a leitura da BEMF induzida no enrolamento desenergizado do motor e seu valor é comparado com o valor médio das tensões dos dois terminais energizados do motor. No momento em que for detectado um cruzamento entre essas tensões, a comutação para a próxima etapa de acionamento deve ser realizada.

Uma vantagem desta técnica é a simplicidade, pois não necessita de nenhum sensor extra no motor. Por outro lado, este princípio só funciona adequadamente a partir de certa faixa de rotação, onde o sinal BEMF é grande o suficiente para ser reconhecido. Dessa forma o motor opera em malha aberta durante a partida, podendo apresentar oscilações até que a velocidade mínima seja atingida. (AKIN; BHARDWAJ, 2015).

A Figura 5 exibe de forma simplificada um circuito elétrico típico utilizado para a leitura da BEMF induzida no terminal do motor. Um divisor de tensão resistivo e um capacitor são utilizados para realizar o condicionamento do sinal.

Figura 5: Circuito de leitura da BEMF induzida no motor *Brushless*



Fonte: Adaptado de: AKIN; BHARDWAJ (2015).

2.2.5. Detecção da posição do rotor - Sensores Hall

Nesta técnica, três Sensores Hall instalados dentro do motor informam ao controlador a posição do rotor segundo uma codificação binária. Este método de detecção apresenta uma resolução de 60º elétricos na identificação da posição do rotor.

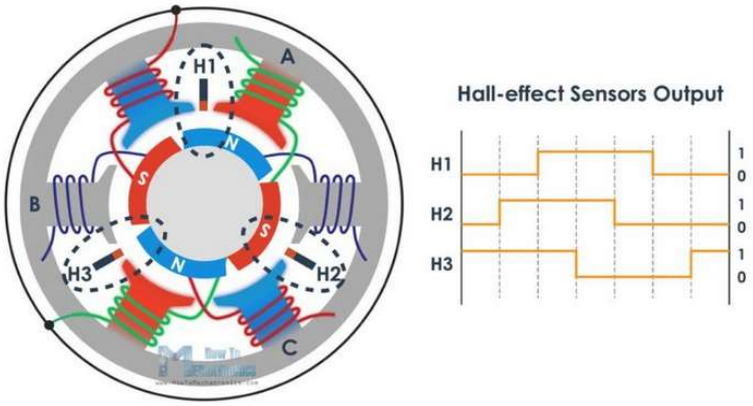
Do ponto de vista de acionamento, através deste método basta verificar se houve mudança de estado lógico em algum sensor, e caso exista, proceder para a respectiva etapa de acionamento.

Esta técnica apresenta como vantagem um controle mais preciso do motor, pois permite a detecção da posição angular do rotor em toda a faixa de rotações, além de também facilitar o uso de dispositivos digitais no controle do motor, pois utiliza sinais lógicos para realimentar o sistema de controle.

Como desvantagem, esta técnica exige que o motor possua Sensores Hall internos, o que nem sempre pode existir no motor *Brushless* disponível.

A Figura 6 exibe à esquerda a localização típica dos Sensores Hall em um motor (entre as ranhuras do estator) e à direita um sinal típico dos Sensores Hall.

Figura 6: Localização dos Sensores Hall e formato de onda associado



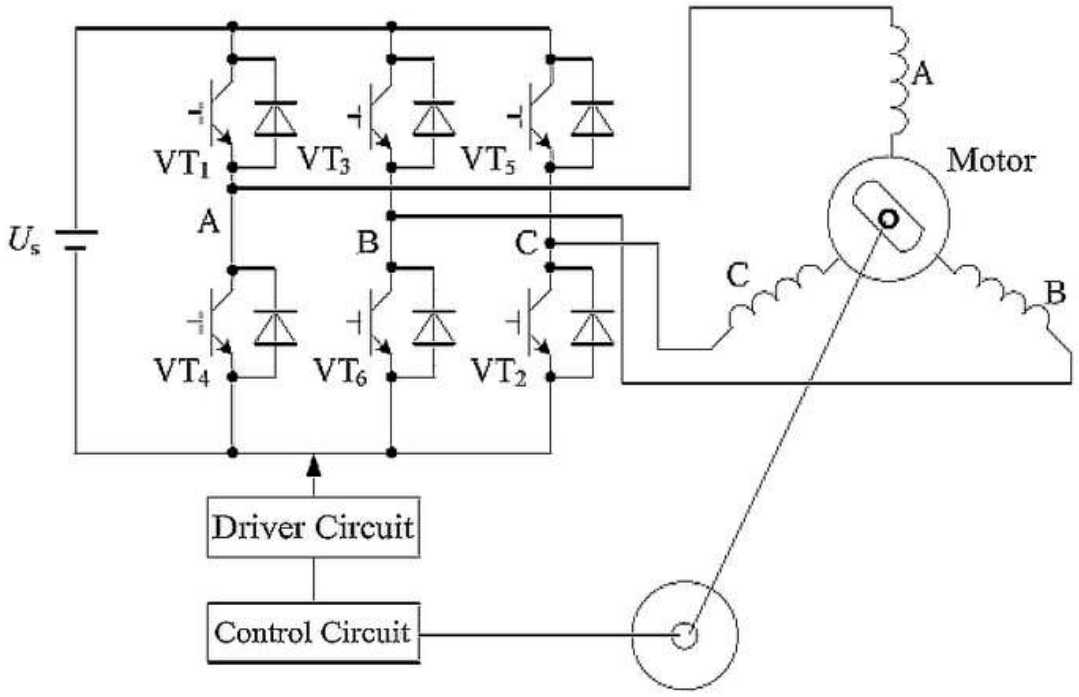
Fonte: RANDRIAKOTONJANAHARY (2019).

2.2.6. Arquitetura para controle de motores *Brushless* DC

A Figura 7 exibe a arquitetura típica utilizada para o controle de motores *Brushless* DC. Nela é possível identificar a forma como cada etapa do sistema está interligada.

Esta arquitetura apresenta os elementos principais em um sistema de controle para tais motores, como o controlador, os *drivers* de acionamento da ponte inversora, a ponte inversora trifásica transistorizada, a bateria, o motor e o laço de realimentação da posição do rotor para o controlador (via Sensores Hall ou por BEMF).

Figura 7: Arquitetura para controle de motores *Brushless* DC



Fonte: YE; HUANG (2018).

2.3. *Field Programmable Gate Array - FPGA*

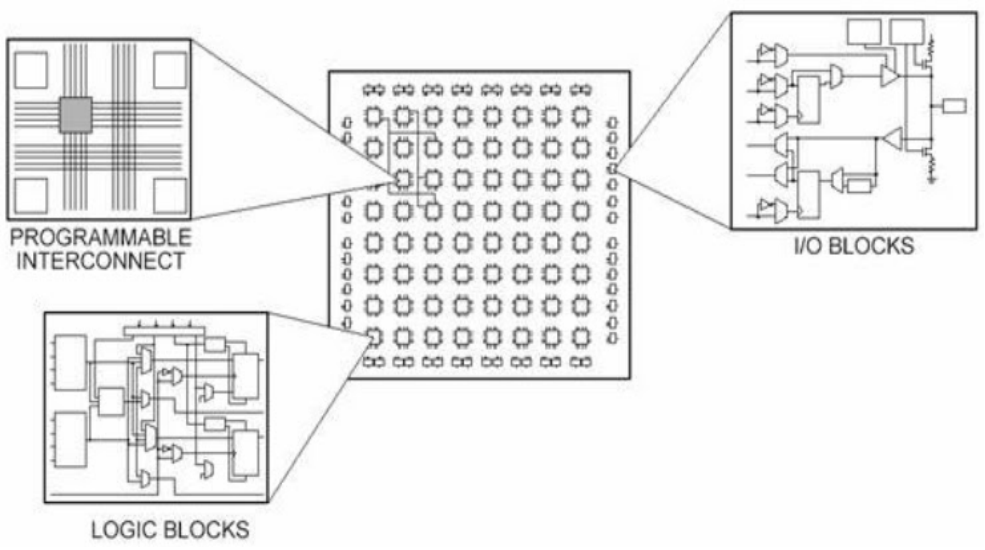
Sigla em inglês para “*Field Programmable Gate Array*”, cuja tradução literal significa “Arranjo de Portas Programável em Campo”.

São dispositivos semicondutores que se baseiam em uma matriz de blocos lógicos configuráveis conectados através de interconexões também configuráveis. Esses dispositivos podem ser programados para uma determinada aplicação ou requisitos de funcionalidade após sua manufatura. Esta característica de programabilidade é o que distingue estes dispositivos dos ASICs (sigla do inglês que em sua tradução significa “Circuito Integrado de Aplicação Específica”), que são componentes personalizados desenvolvidos para tarefas específicas. (XILINX, 2021).

Cada FPGA é constituído por um número finito de blocos lógicos com interconexões programáveis, capazes de implementar um circuito digital reconfigurável. De forma a permitir o acesso ao mundo externo, blocos de entrada/saída também fazem parte da estrutura destes dispositivos. (National Instruments, 2020).

A Figura 8 mostra a arquitetura típica de um dispositivo FPGA, na qual podem ser identificados seus três componentes principais (Blocos Lógicos, Interconexões Programáveis e Blocos de Entrada/Saída).

Figura 8: Arquitetura interna de um FPGA



Fonte: National Instruments (2020).

Em dispositivos FPGA, o número de blocos lógicos configuráveis disponíveis torna-se a unidade básica de medida da capacidade destes componentes. Tais blocos são às vezes definidos como *slices*, outras vezes como células lógicas. Eles são compostos por dois componentes básicos: *Flip-Flops* e *Lookup Tables* (LUTs).

Flip-Flops são registradores de deslocamento binários utilizados para sincronizar e salvar estados lógicos entre ciclos de relógio. A cada borda do sinal de relógio, estes componentes travam seu sinal de saída baseado no valor aplicado em sua entrada até que um novo sinal de relógio seja aplicado.

Lookup Tables são estruturas lógicas que utilizam tabelas-verdade implementadas em pequenas porções de memória RAM com a finalidade de executar funções de lógica combinacional. Uma tabela-verdade é definida como uma lista de valores de saída para cada combinação dos valores de entrada. (National Instruments, 2020).

Dispositivos FPGA possuem ainda outro recurso muito importante a ser considerado em sua especificação, os Blocos de Memória RAM. Esses blocos são úteis para armazenar conjuntos de dados ou transferir valores entre tarefas paralelas. Embora seja possível implementar células de memória utilizando *Flip-Flops*, grandes arranjos implementados desta forma rapidamente se tornam dispendiosos para os recursos lógicos de um FPGA.

Durante os 20 anos iniciais do desenvolvimento dos FPGAs, as linguagens de descrição de hardware VHDL e Verilog se tornaram as principais linguagens utilizadas nos projetos envolvendo estes dispositivos. Ambas são linguagens textuais de baixo nível e sua sintaxe mapeia conexões entre os pinos de entrada/saída e funções lógicas internas que hospedam os algoritmos projetados.

Tais funções lógicas são descritas de forma sequencial, linha por linha. No entanto, devido à natureza paralela de execução das tarefas em um FPGA, a visualização do fluxo de comandos em linguagens textuais se torna mais complexa. (National Instruments, 2020).

3. DESENVOLVIMENTO

O primeiro passo do desenvolvimento deste trabalho é a determinação dos requisitos de projeto que o controlador de motores *Brushless* DC desenvolvido deve possuir. Estes requisitos buscam primariamente orientar as escolhas técnicas realizadas de forma a atingir o objetivo inicialmente proposto.

3.1. REQUISITOS DE PROJETO

Os dois primeiros requisitos de projeto estão relacionados à funcionalidade do controlador desenvolvido e abrangem dois parâmetros fundamentais no acionamento de um motor elétrico, que são o sentido de rotação e a velocidade de operação. O controlador desenvolvido deve ser capaz de mudar o sentido de rotação quando desejado e também deve ser capaz de controlar a velocidade de operação do motor.

O terceiro requisito relaciona a diferença entre a velocidade desejada e a velocidade observada no motor, denominado erro. O controlador desenvolvido deve ser capaz de atingir erro nulo em regime permanente de operação.

3.2. ARQUITETURA DO SISTEMA

De forma a atingir os requisitos de projeto referentes à velocidade de operação do motor (controle de velocidade e erro nulo em regime permanente), foi implementado um sistema de controle em malha fechada com um controlador customizado para a aplicação. Este controlador foi baseado no controlador Proporcional-Integral (PI). (OGATA, 2010).

A Figura 9 exibe um diagrama da arquitetura proposta com o uso deste controlador. Nesse diagrama são identificados os principais sinais utilizados no sistema de controle.

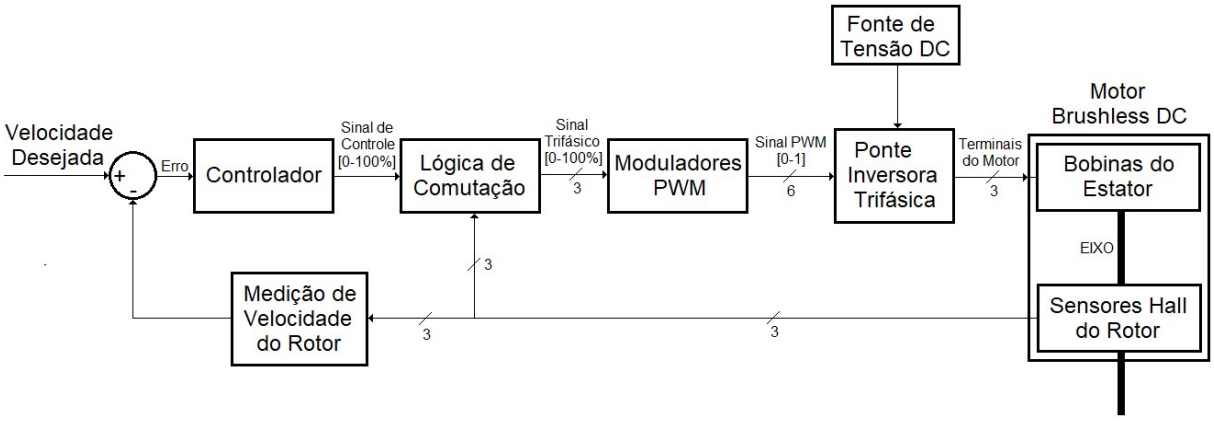
Na entrada de referência do sistema é aplicado o valor de velocidade desejada para o motor, do qual é subtraída a velocidade instantânea do motor para gerar o sinal de erro do sistema. Esse sinal de erro é aplicado ao controlador que efetua os cálculos necessários e gera o sinal de controle, regulando o valor de ciclo ativo da modulação PWM que é aplicada ao motor.

Esse sinal de controle é aplicado ao bloco denominado Lógica de Comutação, responsável por convertê-lo em três variáveis. Cada uma dessas variáveis representa o ciclo ativo da modulação PWM que deve ser aplicado em um enrolamento do motor. Este bloco controla ainda o sentido de rotação do motor, visto que a sequência de acionamento aplicada ao motor é de sua responsabilidade.

As três variáveis de controle da modulação PWM anteriormente geradas são então aplicadas a três moduladores PWM distintos, cujos sinais resultantes são aplicados a uma Ponte Inversora Trifásica, responsável por manipular a corrente elétrica da bateria e efetuar o fornecimento de energia ao motor. Vale notar que este motor utiliza uma ligação elétrica estrela sem neutro, apresentando assim apenas três terminais elétricos.

Para identificação da posição do rotor, três Sensores Hall integrados ao motor são utilizados. Estes sensores oferecem uma resolução de 60º elétricos e indicam o momento ideal para a comutação dos enrolamentos do motor ao bloco Lógica de Comutação. Além disso, os sinais dos Sensores Hall são utilizados para medição de velocidade do rotor, completando o sistema de controle.

Figura 9: Arquitetura do sistema desenvolvido



Fonte: Autor.

3.3. MATERIAIS UTILIZADOS

3.3.1. Motor *Brushless* DC

Para permitir o desenvolvimento e teste do controlador desenvolvido neste trabalho, um motor *Brushless* DC com Sensores Hall integrados foi utilizado. O motor escolhido pertencia a um *Hoverboard* e foi preferido principalmente por se tratar de um conjunto motriz que une motor, roda e pneu em uma única peça.

As características técnicas do conjunto motriz utilizado são: motor de 30 polos magnéticos, tensão elétrica nominal de 36 V, potência elétrica nominal de 350 W, velocidade nominal máxima de 640 RPM @ 36 V e diâmetro externo do pneu de 6,5 polegadas.

A quantidade de polos do motor revela uma relação fundamental utilizada neste trabalho, que é a relação entre frequência elétrica nos enrolamentos e a frequência mecânica do rotor. Como o motor possui 30 polos (15 pares), a frequência elétrica observada em seus enrolamentos será 15 vezes superior à frequência mecânica do rotor. Dessa forma, quando o motor estiver em velocidade máxima (640 RPM = 10,667 Hz), será observada uma onda elétrica com frequência de 160 Hz em seus enrolamentos. Para os Sensores Hall integrados ao motor a mesma relação é válida, sendo observado sob eles uma onda elétrica com frequência de 160 Hz na condição de velocidade máxima do motor.

Como o pneu do conjunto motriz utilizado possui 6,5 polegadas (165,1 mm) de diâmetro externo, seu perímetro resulta em 518,7 mm, ou seja, a cada rotação completa da roda são percorridos aproximadamente 51,9 cm. Considerando que a máxima frequência mecânica de rotação do conjunto motriz é 10,667 Hz, e que cada rotação completa da roda significa um deslocamento de 51,9 cm, então a velocidade máxima que o conjunto motriz consegue atingir é 553,6 cm/s, ou seja, 5,54 m/s, que equivale a 20 km/h.

A partir dessas relações, a velocidade de um veículo que utilize este conjunto motriz pode ser determinada a partir da medição da frequência elétrica dos sinais nos Sensores Hall do motor, onde 160 Hz equivalem a 20 km/h ou 100% da velocidade nominal do motor. De forma proporcional, cada 20 Hz equivalem a 2,5 km/h ou 12,5% da velocidade nominal do motor.

3.3.2. Placa de Desenvolvimento - FPGA

O projeto foi desenvolvido de forma a ser implementado em uma placa de desenvolvimento que será utilizada para a validação do funcionamento. A placa utilizada é produzida pela companhia RUI ZHI YAN FA, marca RZRD, modelo RZ-EasyFPGA, versão A2.2.

Esta placa é baseada em um FPGA Altera Cyclone IV, modelo EP4CE6E22C8N. Um dos atrativos desta placa é a grande diversidade de conexões disponíveis, como por exemplo: VGA, RS232, PS/2, IR e LCD. Esta placa possui ainda uma memória não-volátil para armazenamento do arquivo de configuração do dispositivo, além de diversos pinos localizados próximos do FPGA, que possibilitam acesso direto ao componente, facilitando o processo de desenvolvimento.

O modelo do FPGA utilizado possui um total de 6272 elementos lógicos internos e 92 pinos de entrada/saída. Possui ainda um total de 276480 bits de memória, 30 multiplicadores de 9 bits e 2 PLLs. A frequência máxima de operação para os elementos lógicos internos deste dispositivo é de 250 MHz. Um oscilador baseado em cristal fornece o sinal de relógio de 50 MHz para o dispositivo.

Para a conexão da placa de desenvolvimento ao sistema desenvolvido, foi utilizado o conector de 20 pinos destinado originalmente para o uso de um LCD externo. Este conector foi escolhido, pois possui 9 pinos de entrada/saída (3 para leitura dos Sensores Hall e 6 para acionamento da Ponte Inversora Trifásica) necessários para comunicação do FPGA com o sistema desenvolvido, permitindo ainda acesso às linhas de alimentação de 5 V, 3,3 V e 0 V.

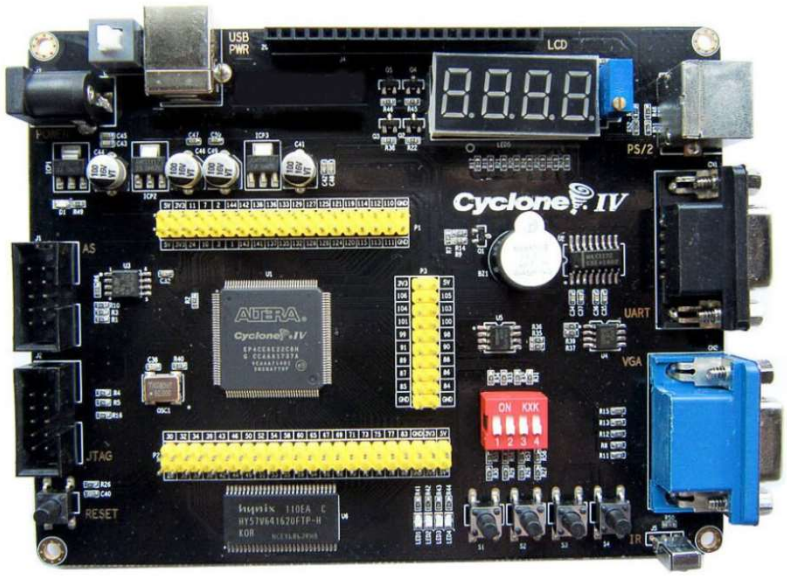
O controle do sistema é exercido pelo usuário através dos quatro botões presentes na parte inferior da placa de desenvolvimento. Ao primeiro e quarto botão foi atribuída a tarefa de frear o motor, isto é, aplicar o valor zero à entrada de referência do sistema, o que por sua vez faz com que o motor pare.

Ao segundo e terceiro botão, foram atribuídas respectivamente as funções de incrementar e decrementar em pequenos passos a entrada de referência do sistema. Dessa forma o usuário pode aumentar ou diminuir a velocidade do motor conforme desejado.

A Figura 10 exibe a placa de desenvolvimento utilizada e os diversos conectores disponíveis para comunicação com o mundo exterior. O conector utilizado para ligar a placa ao sistema localiza-se na parte superior da Figura, onde

há escrito “LCD”. Já os quatro botões utilizados para controle do sistema localizam-se na parte inferior direita da Figura.

Figura 10: Placa de desenvolvimento utilizada



Fonte: RUI ZHI YAN FA (2015).

3.3.3. Baterias

Para fornecer energia elétrica ao sistema desenvolvido, três conjuntos de baterias foram utilizados. Essa divisão foi necessária, pois o sistema exige três valores distintos de tensão elétrica para sua operação, além da necessidade de uma destas tensões ser completamente isolada das demais (sem ponto em comum).

O primeiro conjunto utiliza uma célula Lítio-Íon de 3,6 V com capacidade aproximada de 3600 mAh. Este conjunto possui um circuito elevador que fornece uma tensão elétrica de saída de 5 V em um conector USB Tipo-A fêmea. A finalidade deste conjunto é fornecer alimentação para a placa de desenvolvimento que contém o FPGA. Este conjunto não possui nenhum ponto em comum com as demais baterias, de forma a proteger a placa de desenvolvimento utilizada contra possíveis danos que possam ser causados pelas outras baterias do sistema, por possuírem tensão elétrica mais elevada.

O segundo conjunto utiliza um total de três células Lítio-Íon de 3,6 V com capacidade aproximada de 3600 mAh cada, ligadas em série, de forma a prover uma tensão elétrica nominal de 10,8 V e tensão máxima de 12,6 V quando totalmente carregadas. A finalidade deste conjunto é fornecer alimentação para a Interface de Potência e para a Interface dos Sensores Hall.

O terceiro conjunto é a bateria principal do sistema, possuindo um total de 80 células Lítio-Íon de 3,6 V com capacidade aproximada de 2200 mAh cada. Essas células estão divididas em 4 conjuntos ligados em paralelo. Cada conjunto é composto por um total de 20 células, que são agrupadas inicialmente em 10 pares ligados em série, totalizando 36 V e 4400 mAh cada conjunto. Com todos os seus 4 conjuntos conectados em paralelo, a bateria principal possui 36 V e 17600 mAh de capacidade, totalizando 633,6 Wh de energia armazenada.

3.3.4. Circuitos de Interface

Para efetuar o controle do motor *Brushless* DC a partir de um FPGA, dois circuitos eletrônicos de interface foram desenvolvidos. O primeiro é responsável pelo acionamento dos transistores de potência que fornecem energia ao motor, enquanto que o segundo é responsável pela leitura dos Sensores Hall do motor.

Devido à existência de uma fonte de alimentação DC de alta potência para a alimentação do motor (Bateria principal de 36 V e 17600 mAh), optou-se pela adição de uma barreira de isolação galvânica através do uso de optoacopladores em ambos os circuitos de interface. Tal escolha visa garantir a segurança do usuário, prevenindo contra choques elétricos, além de também prevenir danos à placa de desenvolvimento.

3.3.4.1. Interface de Potência

O objetivo deste circuito é efetuar a conversão de seis sinais lógicos de 3,3 V do FPGA em comandos de acionamento para uma ponte inversora trifásica com seis transistores. De forma complementar, tal circuito deve proporcionar a isolação galvânica entre entrada e saída.

O ponto de partida para o projeto foi a escolha do optoacoplador responsável pela isolação entrada-saída do circuito. O componente escolhido foi o circuito integrado TLP351, pelo fato de possuir em sua saída um driver capaz de garantir que ambos os estados lógicos sejam atingidos em um curto período de tempo, tipicamente menor que 1 μseg.

Para limitar a corrente de entrada aplicada aos optoacopladores, foram utilizados resistores de 330 Ω. Diodos Zener de 8,2 V também foram utilizados, de modo a proteger o componente contra sobretensão e inversão de polaridade na entrada.

O segundo elemento escolhido foi o circuito integrado responsável pelo acionamento dos transistores da ponte inversora trifásica. Devido à velocidade de operação, grande capacidade de corrente de saída e facilidade de acesso, o componente escolhido foi o *Gate Driver* IR2110. Este componente é capaz de efetuar o correto acionamento dos dois transistores de um braço da ponte inversora,

o que acaba reduzindo pela metade a quantidade de circuitos integrados necessários para essa finalidade.

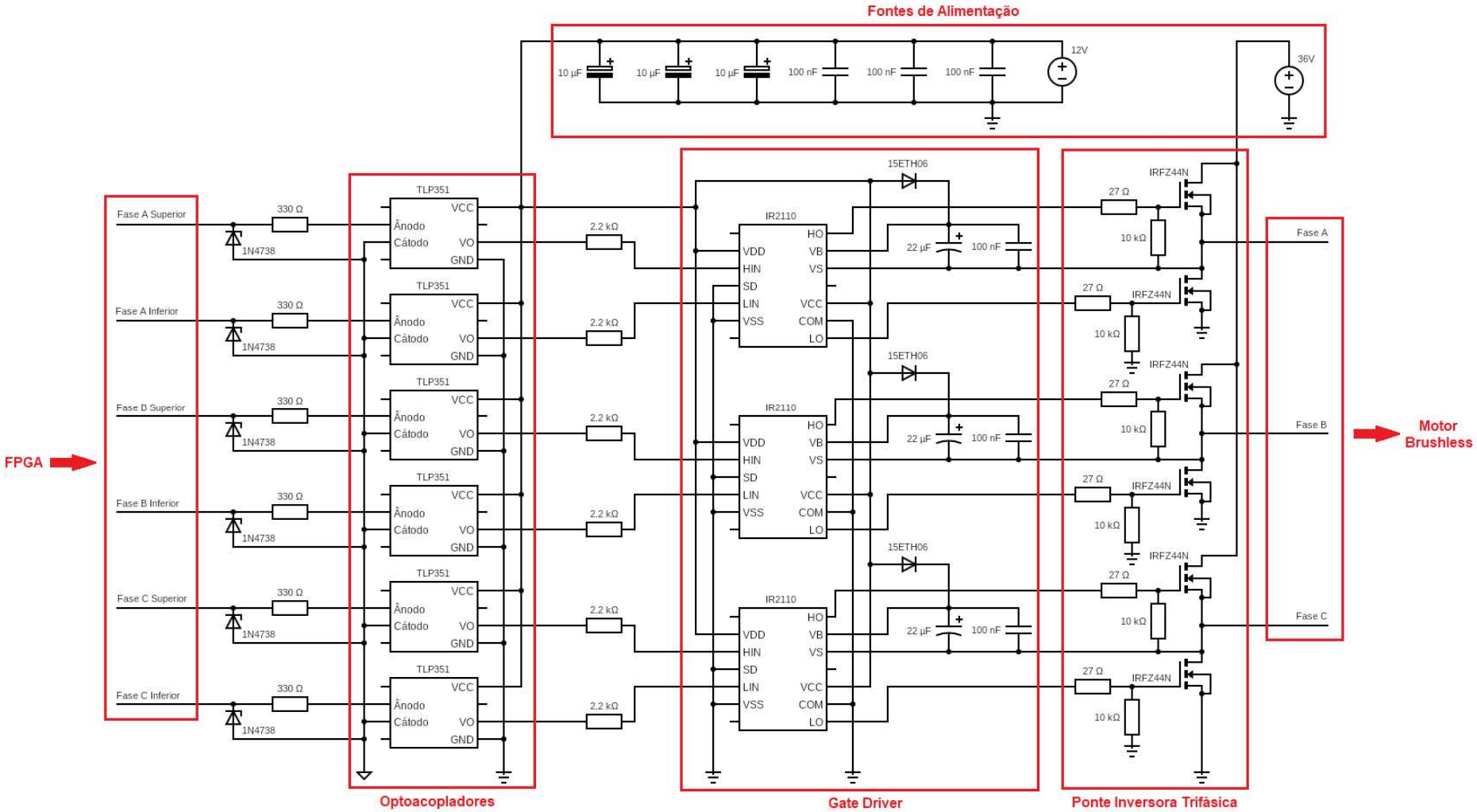
Para o correto funcionamento do circuito integrado IR2110, um diodo de alta velocidade de comutação é necessário para que o capacitor de *bootstrap* se carregue. Dessa forma, foi escolhido o diodo 15ETH06 devido à sua alta capacidade de condução de corrente e velocidade de chaveamento.

O transistor de potência escolhido foi o IRFZ44N, um MOSFET de canal N capaz de suportar 55 V, 49 A e apresentar 17.5 mΩ de resistência de canal em condução. Considerando que o motor utilizado possui corrente nominal de 10 A, o transistor escolhido excede em quase 5 vezes a capacidade de condução de corrente necessária na aplicação.

Para a alimentação dos circuitos integrados de acionamento foi escolhida uma tensão elétrica de 12 V, pois se trata de um valor comercial bastante comum e que atende ao valor mínimo de 10 V requerido pelos componentes do circuito. Para o acionamento do motor foi utilizada uma tensão elétrica de 36 V, compondo assim o circuito principal de alta tensão.

A Figura 11 exibe o diagrama esquemático da interface de potência desenvolvida. Todos os valores e códigos dos componentes utilizados estão indicados na figura.

Figura 11: Circuito da Interface de Potência desenvolvida



Fonte: Autor.

3.3.4.2. Interface dos Sensores Hall

O objetivo deste circuito é efetuar a conversão de três sinais lógicos do tipo coletor aberto, provenientes dos Sensores Hall, em outros três sinais lógicos de 3,3 V para o FPGA. De forma complementar, tal circuito deve proporcionar a isolação galvânica entre entrada e saída.

O ponto de partida para o projeto foi a escolha do optoacoplador responsável pela isolação entrada-saída do circuito. Como este circuito opera em uma velocidade muito menor que a frequência de chaveamento aplicada ao motor, um optoacoplador mais simples foi utilizado. Dessa forma, o componente escolhido foi o circuito integrado 6N135, um optoacoplador de baixo custo e fácil acesso.

O segundo componente escolhido é utilizado como *buffer* de corrente para os sinais dos Sensores Hall. Para tal função foi utilizado um circuito integrado contendo quatro amplificadores operacionais, o TL084.

Este componente foi escolhido pois contém quatro amplificadores operacionais em um único encapsulamento (reduzindo a quantidade de circuitos integrados necessários) e também por possuir uma boa resposta em frequência, que pode chegar às centenas de KHz. Outra vantagem da escolha deste componente é sua alta impedância de entrada, facilitando o acoplamento com os Sensores Hall.

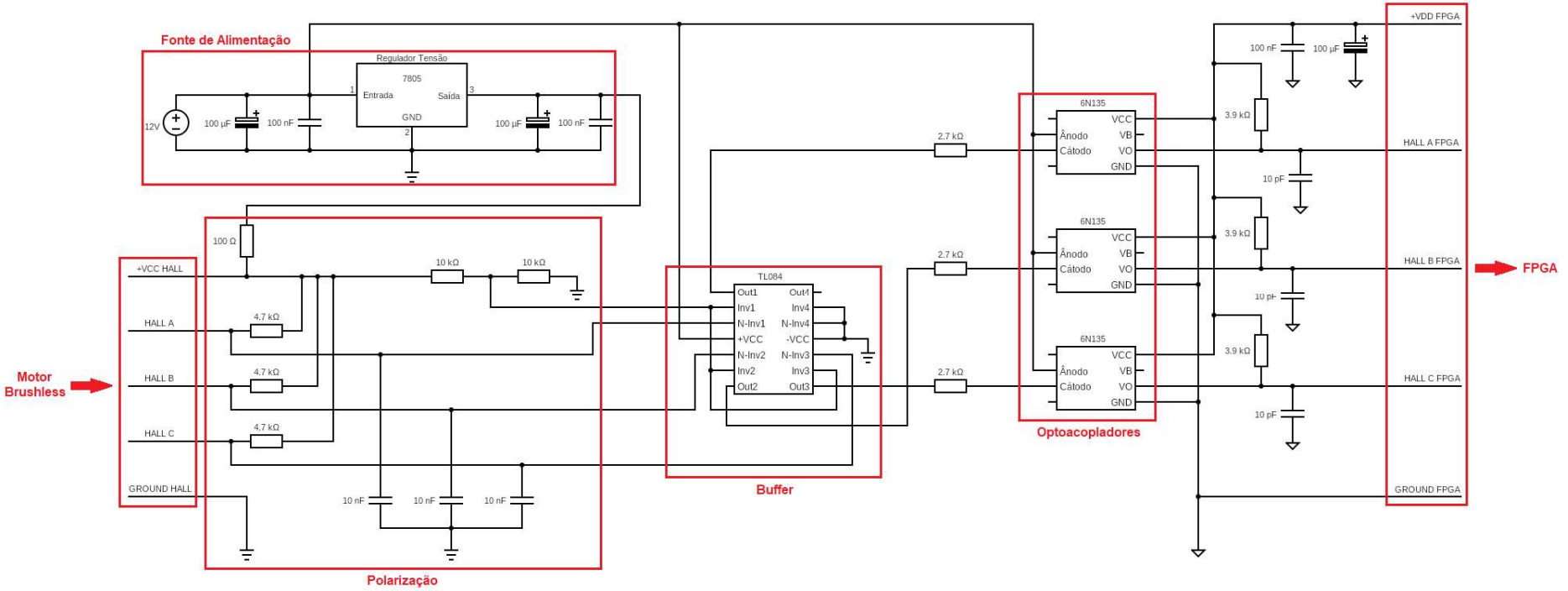
Para a alimentação dos Sensores Hall foi utilizado um regulador de tensão LM7805 de 5 V com um resistor de 100 Ω em série na saída. Este componente tem a função de reduzir a corrente fornecida aos Sensores Hall e também proteger o circuito no caso de um curto-circuito acidental no conector dos sensores do motor.

A alimentação do *buffer* de corrente e dos optoacopladores é realizada diretamente a partir da linha de 12 V. Isto se deve principalmente devido à limitação de tensão mínima de operação do TL084, que é de aproximadamente 7,5 V.

Do ponto de vista eletrônico, o circuito foi idealizado para efetuar a conversão dos sinais lógicos sem inversão. Caso seja aplicado um sinal lógico nível alto na entrada dos Sensores Hall, a saída disponibilizada para o FPGA também apresentará nível lógico alto.

A Figura 12 exibe o diagrama esquemático da interface dos Sensores Hall desenvolvida. Todos os valores e códigos dos componentes utilizados estão indicados na figura.

Figura 12: Circuito da Interface dos Sensores Hall desenvolvida



Fonte: Autor.

3.4. SISTEMA DIGITAL DESENVOLVIDO

O sistema digital do controlador de motores *Brushless* DC desenvolvido neste trabalho foi elaborado a partir da união de uma série de pequenos arquivos escritos em linguagem VHDL. Cada um desses arquivos visa implementar uma função específica, necessária para o funcionamento do controlador. Estes arquivos encontram-se disponíveis no ANEXO 2 deste trabalho.

O programa de computador Quartus Prime 18.1.0 Lite Edition foi utilizado para efetuar a síntese do sistema digital desenvolvido. Para tanto, os arquivos desenvolvidos originalmente em linguagem VHDL foram instanciados sob a forma de blocos funcionais em um arquivo principal de esquemático, disponível no ANEXO 1 deste trabalho. Esse arquivo de esquemático permite que os sinais dos blocos funcionais do sistema sejam interligados através da utilização de conexões representadas por linhas, promovendo assim uma visão de mais alto nível do sistema e facilitando o desenvolvimento.

Os itens subsequentes deste capítulo abordam de forma resumida o funcionamento das principais etapas do sistema desenvolvido. Para informações mais detalhadas a respeito do controlador desenvolvido, recomenda-se a leitura dos arquivos do ANEXO 1 e 2 deste trabalho.

3.4.1. Representação numérica adotada

Para a representação dos valores de frequência elétrica no sistema desenvolvido utilizou-se variáveis do tipo inteiro. Esse tipo de variável foi escolhido pois representa uma economia em termos de recursos lógicos implementados no FPGA quando comparado com variáveis de ponto fixo ou ponto flutuante.

Entretanto, como variáveis do tipo inteiro não conseguem representar números reais (com casas decimais), isso resulta em uma menor resolução nos cálculos efetuados quando comparado com variáveis de ponto fixo ou flutuante. De forma a contornar parte desse problema, foi utilizada uma representação similar ao ponto fixo, de forma que o último dígito do número inteiro equivale à primeira casa decimal do número real. Por exemplo: o número real 123,4 equivale a 1234 na representação numérica adotada. Assim, é possível obter uma maior resolução na

representação das variáveis mantendo a simplicidade de cálculo proporcionada pelo uso de variáveis do tipo inteiro.

Este tipo de representação numérica foi adotada em todas as variáveis do sistema desenvolvido que estão relacionadas a frequência elétrica. Também adotou-se essa representação para as constantes utilizadas nos cálculos do controlador.

3.4.2. Ajuste da Velocidade Desejada

Dado que o objetivo do sistema projetado é controlar a velocidade de operação de um motor *Brushless* DC, e que tal velocidade apresenta uma relação proporcional com a frequência elétrica observada nos Sensores Hall do motor, conforme apresentado na seção 3.3.1, o objetivo inicial do projeto pode ser alcançado através da excitação controlada do motor e monitoramento da frequência elétrica dos sinais dos Sensores Hall. Dessa forma, o sistema elaborado utiliza internamente parâmetros de frequência elétrica para controlar a velocidade de operação do motor.

Conforme apresentado na seção 3.3.1, a velocidade máxima alcançada pelo conjunto motriz utilizado é de 20 km/h, o que resulta em uma frequência elétrica de 160 Hz nos sinais dos Sensores Hall do motor. Essa seção ainda estabelece que cada 20 Hz de frequência elétrica observada nestes sensores equivalem a 2,5 km/h.

Dessa forma, o controle da velocidade de operação do motor é realizado informando ao controlador o valor da frequência elétrica que se deseja observar nos Sensores Hall (de 0 a 160 Hz).

Para informar ao sistema a frequência elétrica desejada de operação do motor (determinando assim sua velocidade), um código em linguagem VHDL denominado BUTTON.vhd foi elaborado de forma a monitorar os quatro botões da placa de desenvolvimento e alterar o valor de frequência desejada no motor conforme necessário. Para tanto, dois dos quatro botões efetuam a parada completa do motor, aplicando o valor zero como frequência desejada. Os outros dois botões efetuam respectivamente incrementos ou decrementos de 20,0 Hz no parâmetro frequência desejada a cada vez que o botão é pressionado.

Assim, um total de 8 incrementos de 20,0 Hz podem ser aplicados ao sistema até que este atinja sua frequência máxima de operação de 160,0 Hz. Em termos de velocidade, cada um desses oito incrementos equivale a um acréscimo de

2,5 km/h, totalizando assim uma velocidade máxima de 20 km/h. Em termos percentuais, cada um desses oito incrementos equivale a um acréscimo de 12,5% na velocidade de operação do motor.

A Figura 13 exibe o trecho principal do arquivo BUTTON.vhd desenvolvido, responsável por efetuar o ajuste da frequência elétrica desejada de operação do motor, determinando assim sua velocidade de operação.

Figura 13: Trecho do código de ajuste da velocidade do motor

```
PROCESS (clk,reset)
BEGIN
   IF (clk'EVENT AND clk = '1') THEN
      IF (reset = '0') THEN
         sp_reg_calc <= 0;
      ELSE
         IF (smp_reg = '0') AND (smp = '1') THEN
            IF (but_11_reg = '1') AND (but_1_reg = '0') THEN
               sp_reg_calc <= 0;
            ELSE
               IF (but_22_reg = '1') AND (but_2_reg = '0') THEN
                  IF (sp_reg_calc > 199) THEN
                     sp_reg_calc <= sp_reg_calc - 200;
                  ELSE
                     sp_reg_calc <= sp_reg_calc;
                  END IF;
               ELSE
                  IF (but_33_reg = '1') AND (but_3_reg = '0') THEN
                     IF (sp_reg_calc < 1401) THEN
                        sp_reg_calc <= sp_reg_calc + 200;
                     ELSE
                        sp_reg_calc <= sp_reg_calc;
                     END IF;
                  ELSE
                     IF (but_44_reg = '1') AND (but_4_reg = '0') THEN
                        sp_reg_calc <= 0;
                     ELSE
                        sp_reg_calc <= sp_reg_calc;
                     END IF;
                  END IF;
               END IF;
            ELSE
               sp_reg_calc <= sp_reg_calc;
            END IF;
         END IF;
      END IF;
END PROCESS;
```

Fonte: Autor.

3.4.3. Medição da Velocidade do Motor

A velocidade de operação do motor é medida através da frequência elétrica observada nos sinais dos Sensores Hall. Para tal medição ser executada, um código em linguagem VHDL denominado COUNTER_1US.vhd foi elaborado.

Esse código implementa um contador com *reset* que efetua incrementos de uma unidade em seu valor de contagem a cada 1 µs. Seu objetivo é medir o período da onda dos Sensores Hall e dessa forma determinar a frequência elétrica dos sinais nestes sensores.

A Figura 14 exibe o trecho principal do arquivo COUNTER_1US.vhd que implementa esse contador.

Figura 14: Trecho do código do contador implementado

```
PROCESS (clk,reset)
BEGIN
   IF (clk'EVENT AND clk = '1') THEN
      IF (reset = '0') THEN
         sub_counter_value_reg <= 0;
         counter_value_reg <= 0;
         counter_value_reg_1 <= 0;
         counter_value_reg_2 <= 0;
         counter_value_reg_3 <= 0;
         counter_value_reg_4 <= 0;
         counter_reset_reg <= '0';
      ELSE
         counter_reset_reg <= counter_reset;
            IF (counter_reset_reg = '0') AND (counter_reset = '0')  THEN
               IF (sub_counter_value_reg < (CLOCK_MHZ - 1)) THEN
                  sub_counter_value_reg <= sub_counter_value_reg + 1;
               ELSE
                  sub_counter_value_reg <= 0;
                  IF (counter_value_reg < 249000) THEN
                     counter_value_reg <= counter_value_reg + 1;
                  ELSE
                     counter_value_reg <= counter_value_reg;
                  END IF;
               END IF;
            END IF;
            IF (counter_reset_reg = '0') AND (counter_reset = '1')  THEN
               counter_value_reg_1 <= counter_value_reg;
               counter_value_reg_2 <= counter_value_reg_1;
               counter_value_reg_3 <= counter_value_reg_2;
               counter_value_reg_4 <= counter_value_reg_3;
            END IF;
            IF (counter_reset_reg = '1') AND (counter_reset = '1')  THEN
               sub_counter_value_reg <= 0;
               counter_value_reg <= 0;
            END IF;
            IF (counter_reset_reg = '1') AND (counter_reset = '0')  THEN
               sub_counter_value_reg <= 0;
               counter_value_reg <= 0;
            END IF;
      END IF;
   END IF;
END PROCESS;
```

Fonte: Autor.

Para realizar a conversão do período medido em um valor de frequência, um código em linguagem VHDL denominado PERIOD_TO_FREQ.vhd foi elaborado. Esse código recebe o valor do período medido e disponibiliza dois valores de saída denominados *dividend* e *divisor*, que são utilizados posteriormente em uma operação de divisão, dando origem ao valor de frequência elétrica observada dos Sensores Hall.

A Figura 15 exibe o trecho principal do arquivo PERIOD_TO_FREQ.vhd que implementa a função de conversão.

Figura 15: Trecho do código da conversão período-frequência

```
PROCESS (clk)
BEGIN
   IF (clk'EVENT AND clk = '1') THEN
      IF (reset = '0') THEN
         dividend <= std_logic_vector(to_unsigned(0, dividend'length));
         divisor <= std_logic_vector(to_unsigned(1, divisor'length));
         period_reg <= 0;
      ELSE
         period_reg <= to_integer(unsigned(period));
         IF (period_reg > 2940) THEN
            IF (period_reg > 248000) THEN
               dividend <= std_logic_vector(to_unsigned(0, dividend'length));
               divisor <= std_logic_vector(to_unsigned(1, divisor'length));
            ELSE
               dividend <= std_logic_vector(to_unsigned(10000000, dividend'length));
               divisor <= std_logic_vector(to_unsigned((period_reg * 2), divisor'length));
            END IF;
         ELSE
            dividend <= std_logic_vector(to_unsigned(10000000, dividend'length));
            divisor <= std_logic_vector(to_unsigned(5880, divisor'length));
         END IF;
      END IF;
   END IF;
END PROCESS;
```

Fonte: Autor.

3.4.4. Operação de Divisão

Para realizar operações matemáticas de divisão no projeto, um código em linguagem VHDL denominado DIVISION.vhd foi elaborado. Sua tarefa é realizar a divisão de dois números inteiros sem sinal.

A primeira tentativa de implementação desta operação foi executada através do uso do operador divisão “/” disponível na linguagem VHDL. Nesta tentativa, constatou-se um alto uso de elementos lógicos do FPGA (aproximadamente 750) e frequência máxima de operação da lógica implementada de aproximadamente 15 MHz.

Para otimizar essa operação de divisão, o algoritmo de divisão sem restauração de números inteiros sem sinal foi implementado. Este algoritmo foi

escolhido pois utiliza operações matemáticas mais simples quando comparado com o algoritmo que utiliza restauração. (Tutorialspoint, 2018).

Após a implementação deste algoritmo, verificou-se que foram utilizados 181 elementos lógicos (redução de 4 vezes) e a frequência máxima de operação passou para 124 MHz (aumento de 8 vezes). Com base nesses resultados, o algoritmo de divisão sem restauração se mostrou mais eficiente que o uso do operador divisão diretamente no código VHDL, e portanto, essa foi a solução escolhida para executar a operação de divisão.

3.4.5. Controlador

Para efetuar o controle da velocidade de operação do motor foi utilizado um controlador customizado para a aplicação, apresentando funcionamento similar a um controlador Proporcional-Integral (PI), pois um dos requisitos do projeto é obter erro nulo em regime permanente. Esse controlador foi implementado a partir de um código escrito em linguagem VHDL denominado CONTROL_LOOP.vhd. Sua função é determinar o valor de ciclo ativo que deve ser aplicado ao motor baseado na diferença existente entre a frequência desejada de operação e a frequência atual do motor.

O código do controlador deste trabalho foi implementado a partir do modelo de algoritmo do controlador PID proposto por CONCEIÇÃO, BECCARO e JUSTO (2020), onde trechos deste algoritmo foram adaptados de forma a atender às necessidades do projeto.

Os testes realizados durante o desenvolvimento deste código permitiram identificar a necessidade de um comportamento assimétrico por parte do controlador, pois deseja-se uma aceleração suave do veículo, bem como a capacidade de realizar a frenagem do veículo de forma rápida no caso de uma emergência. Dessa forma, o controlador customizado foi desenvolvido para apresentar um comportamento assimétrico, sendo capaz de realizar acelerações de forma suave e efetuar frenagens de forma rápida.

Conforme explicado na seção 3.4.1, os valores de frequência elétrica utilizados no sistema são representados através de variáveis do tipo inteiro onde seu último dígito equivale à primeira casa decimal do valor real da variável. Do ponto de vista numérico, o valor representado nas variáveis do tipo inteiro é 10 vezes superior

ao valor real da variável. Esse formato de representação também é utilizado na definição das constantes de ganho do controlador.

A consequência disso é que o sinal de saída do controlador está escalonado por um fator de 100 vezes em relação ao valor original das variáveis, uma vez que o valor das frequências aplicadas em suas entradas está escalonado em 10 vezes e o valor de suas constantes de ganho também estão escalonadas em 10 vezes. Dessa forma, o sinal de saída do controlador deve ser dividido por 100 para somente então ser utilizado para controlar o ciclo ativo da modulação PWM aplicada ao motor.

O controle da modulação PWM aplicada ao motor foi idealizado para ser realizado através de uma variável cuja faixa de operação seja de 0 a 100. Assim, como o sinal de saída do controlador está escalonado 100 vezes em relação ao valor real, sua faixa de operação deve se situar entre 0 e 10000.

Para evitar que o controlador desenvolvido atinja a saturação do termo integral, efeito conhecido como *Windup*, o valor do termo integral foi limitado entre -10001 e 10001. De forma similar, para evitar que o controlador produza um sinal fora da faixa idealizada de operação, seu sinal de saída foi limitado entre 0 e 10000.

A Figura 16 exibe o trecho principal do arquivo CONTROL_LOOP.vhd que implementa o controlador utilizado.

Figura 16: Trecho do código do controlador customizado

```
E_reg <= (sp_reg - pv_reg);
IF (smp_reg = '0') AND (smp = '1') THEN
   IF (E_reg > 0) THEN
      P_reg <= (E_reg * 8);
      IF (I_reg > 10000) THEN
         I_reg <= 10001;
      ELSE
         I_reg <= I_reg + (E_reg / 32);
      END IF;
   ELSE
      P_reg <= 0;
      IF (E_reg < 0) THEN
         IF (I_reg < -10000) THEN
            I_reg <= -10001;
         ELSE
            I_reg <= I_reg + (E_reg / 4);
         END IF;
      ELSE
         I_reg <= I_reg;
      END IF;
   END IF;
END IF;
mv_calc_reg <= (P_reg + I_reg);
IF (mv_calc_reg < 0) THEN
   mv_reg <= 0;
ELSE
   IF (mv_calc_reg > 10000) THEN
      mv_reg <= 10000;
   ELSE
      mv_reg <= mv_calc_reg;
   END IF;
END IF;
```

Fonte: Autor.

3.4.6. Lógica de Comutação

O controle da sequência de comutação das bobinas do motor *Brushless* é realizado através de um código escrito em linguagem VHDL denominado COMMUTATION.vhd. Sua função é determinar os terminais do controlador que devem ser acionados através do monitoramento dos sinais dos Sensores Hall.

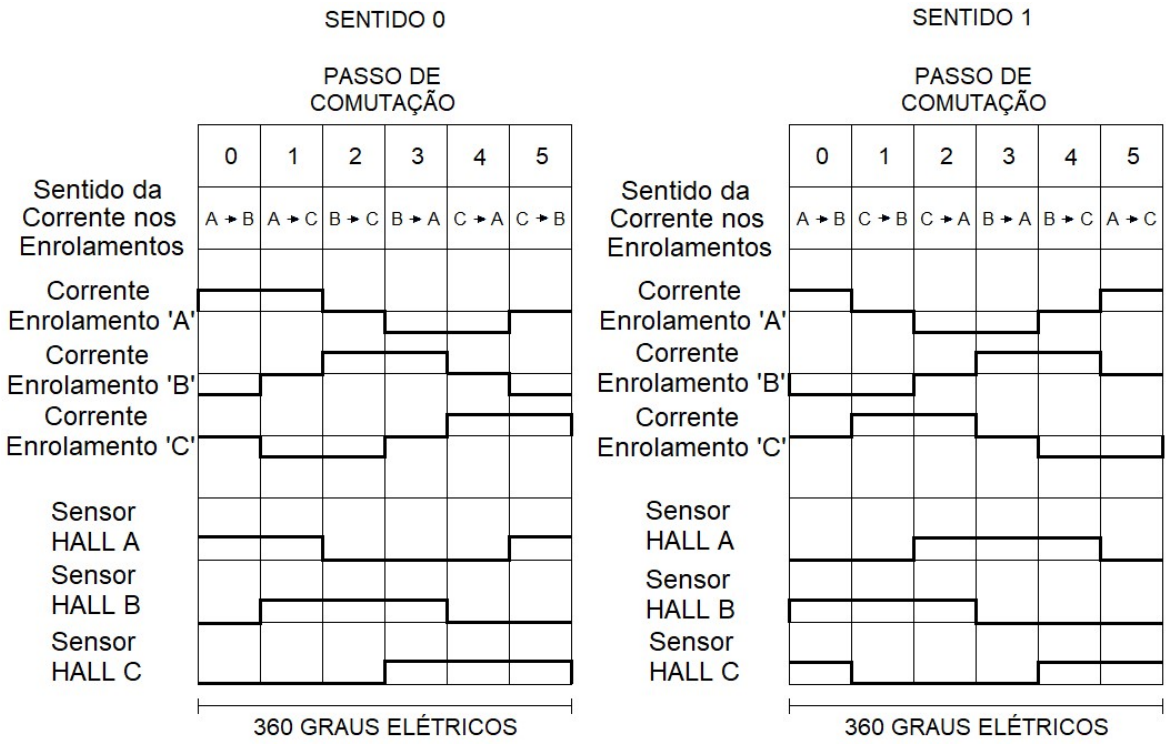
Esse bloco basicamente efetua uma tarefa de decodificação, pois os valores dos três Sensores Hall do motor e do sinal de sentido de rotação desejado indicam os terminais a serem energizados e sua polaridade. Ao total são gerados seis sinais de saída, sendo cada par responsável por controlar a modulação de um terminal de saída do controlador desenvolvido.

Uma vez determinada a polaridade e os terminais que devem ser energizados, este bloco aplica o valor da variável de controle do PWM ao modulador que controla o terminal que deve ser energizado positivamente, aplicando o valor zero ao modulador que controla o terminal que deve ser energizado negativamente (dessa forma o terminal polarizado negativamente fica ligado diretamente ao negativo da fonte de alimentação). Durante esse período, o terminal do controlador que não necessita ser energizado é mantido em estado de alta impedância. Dessa forma, somente há a circulação de corrente elétrica pelo motor *Brushless* do terminal polarizado positivamente para o polarizado negativamente.

Esse tipo de modulação, presente apenas no terminal positivamente polarizado, foi escolhida pois neste projeto é utilizado um circuito integrado *Gate Driver* baseado em capacitores de *bootstrap*. Isso faz com que exista um tempo máximo que o transistor de saída possa permanecer ligado antes que a carga do capacitor acabe. Carga esta que é renovada durante o chaveamento do componente para o estado de desligado.

A Figura 17 exibe dois diagramas que relacionam o sentido de rotação do motor, os sinais dos Sensores Hall e os sinais aplicados aos terminais do controlador para cada passo de comutação.

Figura 17: Sequência de comutação aplicada ao motor



Fonte: Autor.

3.4.7. Modulador PWM

O modulador PWM utilizado neste projeto foi desenvolvido em um código escrito em linguagem VHDL denominado PWM_DEAD_TIME.vhd. Ao todo foram utilizados três moduladores nesse projeto, todos idênticos. A função de cada modulador é controlar a razão cíclica dos sinais que efetuam o acionamento de cada braço da ponte inversora trifásica.

Seu funcionamento é baseado em um contador interno que incrementa uma variável a cada ciclo de *clock* do FPGA. Quando o valor máximo de contagem é atingido, o contador reinicia a contagem. A modulação PWM pode ser obtida através da comparação de um valor modulante com o valor do contador. Quando o valor do contador é menor que o valor modulante, a saída que controla o transistor superior da ponte inversora recebe nível lógico alto, e quando o valor do contador é maior que o valor modulante, a saída que controla o transistor inferior da ponte inversora recebe nível lógico alto.

Como os transistores da ponte inversora não respondem instantaneamente aos comandos de ligar e desligar, apresentando certo atraso na resposta, foi incluído

no modulador um parâmetro denominado *dead time*, que aplica um atraso programável entre a ativação das saídas do modulador. Dessa forma, os transistores da ponte inversora possuem um período de tempo destinado exclusivamente para sua comutação, evitando assim que ambos os transistores da ponte inversora conduzam corrente ao mesmo tempo e causem um curto-circuito temporário na fonte de alimentação.

3.4.8. Recursos lógicos utilizados e frequência máxima de operação

De acordo com o programa de computador Quartus Prime 18.1.0 Lite Edition, utilizado na síntese do projeto, o sistema digital desenvolvido utiliza um total de 1500 elementos lógicos em sua implementação, o que representa cerca de 24% do total disponível no modelo de FPGA utilizado.

A Tabela 2 exibe um resumo dos recursos totais utilizados na implementação do sistema digital desenvolvido.

Tabela 2: Recursos totais utilizados - FPGA

	Recursos Utilizados	Percentual de uso
Elementos lógicos	1500	24%
Registradores	854	Não informado
Pinos de Entrada/Saída	21	23%
Bits de memória	0	0%
Multiplicadores de 9 bits	0	0%
PLLs	0	0%

Fonte: Autor.

A respeito da frequência máxima de operação do sistema digital desenvolvido, a ferramenta computacional utilizada informa que, segundo o modelo para o pior caso (85 ºC) o sistema é capaz de operar até uma frequência de 96,98 MHz.

3.5. PROTÓTIPO DESENVOLVIDO

De forma a validar o funcionamento do controlador desenvolvido neste trabalho, um protótipo de veículo elétrico foi elaborado. Para tanto, utilizou-se como base do protótipo um patinete de alumínio marca Sundown.

O primeiro passo na confecção do protótipo foi a realização de algumas modificações mecânicas com o objetivo de instalar o conjunto motriz na parte traseira do veículo e estender o garfo dianteiro para a instalação de uma roda maior, compatível com o tamanho do conjunto motriz utilizado. Essas modificações foram realizadas através da confecção de peças customizadas em aço, que foram aparafusadas à estrutura original de alumínio.

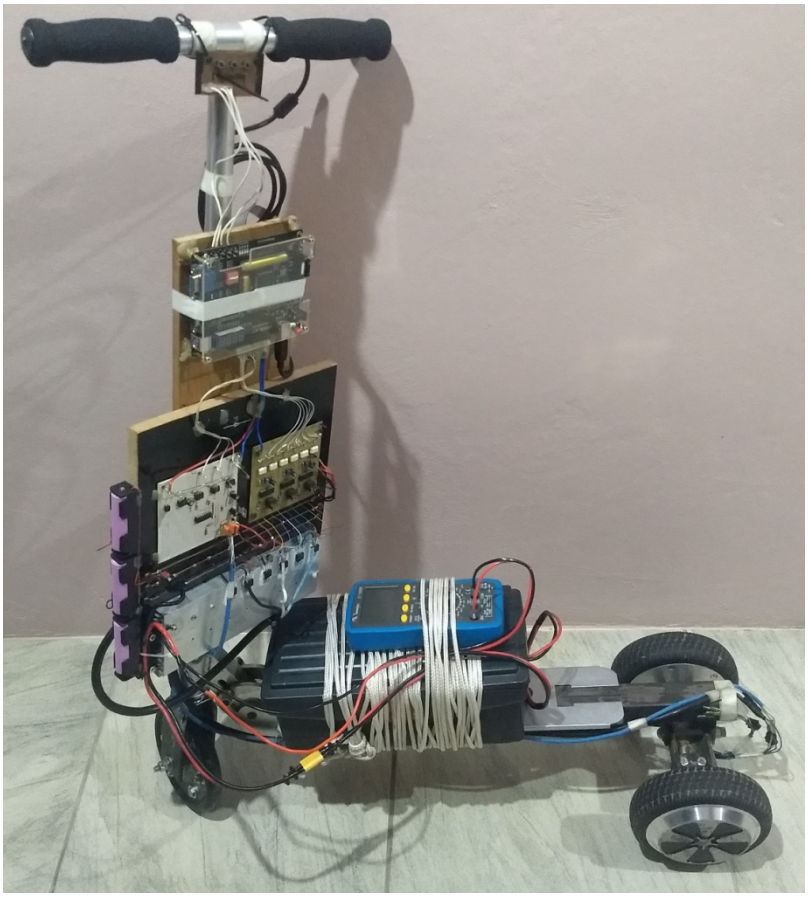
A bateria principal do sistema foi instalada no centro do veículo, mais precisamente sobre a plataforma que liga o eixo dianteiro ao traseiro. Em cima dela foi instalado um multímetro digital que é utilizado para monitorar a corrente elétrica da bateria principal do sistema.

Na parte frontal do veículo foi instalada uma estrutura de madeira que serve de suporte para o controlador desenvolvido. Essa estrutura contém os seguintes itens: Ponte Inversora Trifásica; Interface de Potência; Interface dos Sensores Hall; Bateria de 10,8 V e a Placa de Desenvolvimento - FPGA.

Para o ajuste da velocidade desejada do veículo foi instalada uma pequena placa no guidão que serve de extensão para os botões da placa de desenvolvimento. O primeiro botão incrementa a velocidade desejada, o segundo decrementa a velocidade desejada e o terceiro efetua a frenagem do veículo.

A Figura 18 exibe o protótipo de veículo elétrico desenvolvido que utiliza o controlador de motores *Brushless* DC deste trabalho para controlar o motor traseiro esquerdo.

Figura 18: Protótipo de veículo elétrico desenvolvido



Fonte: Autor.

4. EXPERIMENTOS E RESULTADOS

4.1. PARAMETRIZAÇÃO DO CONTROLADOR

De forma a atingir um regime de funcionamento adequado ao tipo de motor e características da aplicação, um processo de parametrização do controlador foi realizado. Esse processo tem por objetivo ajustar as características de resposta do controlador para torná-las compatíveis com a aplicação final, nesse caso o veículo elétrico.

Na primeira etapa da parametrização as frequências de atualização e modulação dos sinais do sistema foram ajustadas. Para os sinais dos Sensores Hall foi adotada uma frequência de atualização de 1666,66 Hz (mais de 10 vezes superior à maior frequência destes sinais, que é 160 Hz, de forma a evitar *aliasing*). Para os botões de ajuste da velocidade desejada foi escolhida uma frequência de atualização de 10 Hz (suficientemente rápida para a detecção de uma tecla ser pressionada). Para o controlador customizado foi escolhida uma frequência de atualização de 166,66 Hz (como a velocidade do veículo elétrico testado apresenta uma variação lenta, na ordem de segundos, esta frequência de atualização é suficientemente rápida para realizar o controle do motor). Já para a modulação PWM aplicada ao motor foi utilizada uma frequência de 25 KHz (dessa forma a produção de ruído audível pelo motor é minimizada).

Na segunda etapa da parametrização as constantes de ganho do controlador customizado desenvolvido foram ajustadas. Para tanto, uma metodologia alternativa e simplificada foi adotada de forma a permitir a realização de testes práticos com o veículo no curto período de tempo disponível para os testes.

A cada teste realizado, apenas um dos valores de ganho do controlador era alterado, de forma a identificar o impacto deste parâmetro no sistema. Essa metodologia foi repetida várias vezes até que todos os valores de ganho do controlador foram ajustados de forma a obter a melhor resposta do veículo elétrico utilizado para testes.

4.2. INSTRUMENTOS DE MEDIDA UTILIZADOS

Para a realização das medidas elétricas nos experimentos deste trabalho foram utilizados ao todo três equipamentos: dois multímetros digitais e um osciloscópio portátil. As marcas, modelos e escalas utilizadas são informadas nos parágrafos a seguir.

Um multímetro digital marca Minipa, modelo ET-2082C, foi utilizado como amperímetro na escala de 20 A para monitorar a corrente elétrica da bateria principal do sistema.

Um segundo multímetro digital, marca Minipa, modelo ET-1002, foi utilizado como voltímetro na escala de 200 V para monitorar a tensão elétrica da bateria principal do sistema.

Um osciloscópio portátil marca Hantek, modelo 2D72, de 2 canais e 70 MHz foi utilizado para monitorar os sinais dos Sensores Hall do motor.

4.3. EXPERIMENTOS SEM CARGA

Os experimentos desta seção foram denominados “experimentos sem carga” pois as medições executadas foram realizadas com o motor *Brushless* DC operando livre, sem contato com o solo ou qualquer outro objeto, sendo assim sem carga mecânica externa aplicada em seu eixo.

4.3.1. Velocidade em regime permanente

O objetivo deste experimento é avaliar o comportamento do sistema quando operando em regime permanente sem carga mecânica aplicada.

Para testar o controlador foram escolhidos quatro diferentes valores de velocidade desejada para o sistema: 25%, 50%, 75% e 100% da velocidade nominal do motor. Esses valores correspondem respectivamente a sinais de 40 Hz, 80 Hz, 120 Hz e 160 Hz nos Sensores Hall do motor, conforme explicado na seção 3.3.1. Após a aplicação de cada valor de referência, aguardou-se aproximadamente 10 segundos como forma de garantir que o estado de regime permanente seja atingido, momento no qual foram efetuadas medidas de frequência no sinal dos Sensores Hall do motor.

Essas medidas foram realizadas na saída da Interface dos Sensores Hall, em apenas um dos três sinais, pois os outros dois sinais apresentam exatamente a mesma frequência de operação, exceto por uma diferença de fase de 120º elétricos.

A Tabela 3 exibe os dados obtidos no experimento.

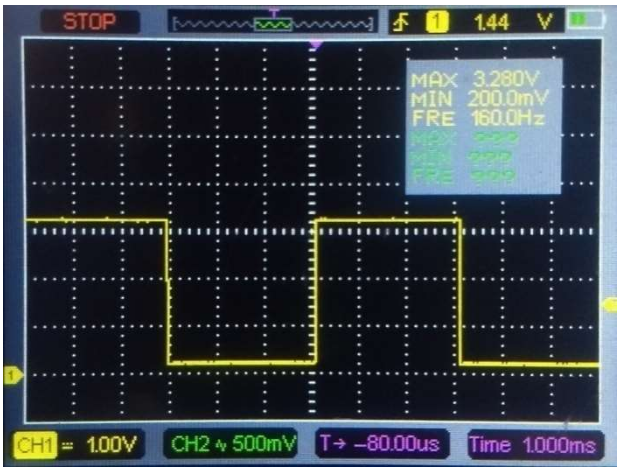
Tabela 3: Valores de frequência elétrica obtidos no experimento

Velocidade do Motor [%]	Frequência do sinal dos Sensores Hall [Hz]
0	0
25	40
50	80
75	120
100	160

Fonte: Autor.

A Figura 19 exibe um exemplo do formato da onda de tensão elétrica produzida pelo Sensor Hall.

Figura 19: Exemplo do Sinal do Sensor Hall - 100% da velocidade



Fonte: Autor.

A partir das medidas realizadas é possível concluir que o controlador desenvolvido consegue controlar a velocidade de operação do motor *Brushless* e ainda atingir erro nulo em regime permanente, indicando que o controlador customizado desempenha seu papel satisfatoriamente.

4.3.2. Sentido de rotação

Um segundo experimento foi realizado para avaliar o comportamento do controlador quando operando no sentido oposto de rotação do motor. O experimento foi conduzido seguindo a mesma metodologia do item 4.3.1.

Os resultados obtidos nesse experimento foram idênticos aos obtidos no item anterior, comprovando que o controlador consegue controlar a velocidade do motor *Brushless* independentemente do sentido de rotação. Dessa forma, o último requisito do projeto foi satisfatoriamente atingido (controle do sentido de rotação).

4.3.3. Corrente elétrica em regime permanente

O objetivo deste experimento é avaliar o consumo de corrente elétrica da bateria principal quando o sistema opera em regime permanente sem carga mecânica aplicada.

Para a realização desse experimento foram escolhidos oito diferentes valores de referência para o sistema: 12,5%, 25%, 37,5%, 50%, 62,5%, 75%, 87,5% e 100% da velocidade nominal do motor. Após a aplicação de cada valor de referência, aguardou-se aproximadamente 10 segundos como forma a garantir que o estado de regime permanente seja atingido, momento no qual foram efetuadas as medidas de corrente elétrica na bateria principal.

O experimento foi repetido um total de três vezes e ao seu término os valores médios foram calculados através de média aritmética simples. A tensão elétrica da bateria principal foi monitorada durante a realização do experimento e se manteve constante em 39,5 V.

A Tabela 4 exibe os dados obtidos nas três repetições do experimento juntamente com a média aritmética calculada.

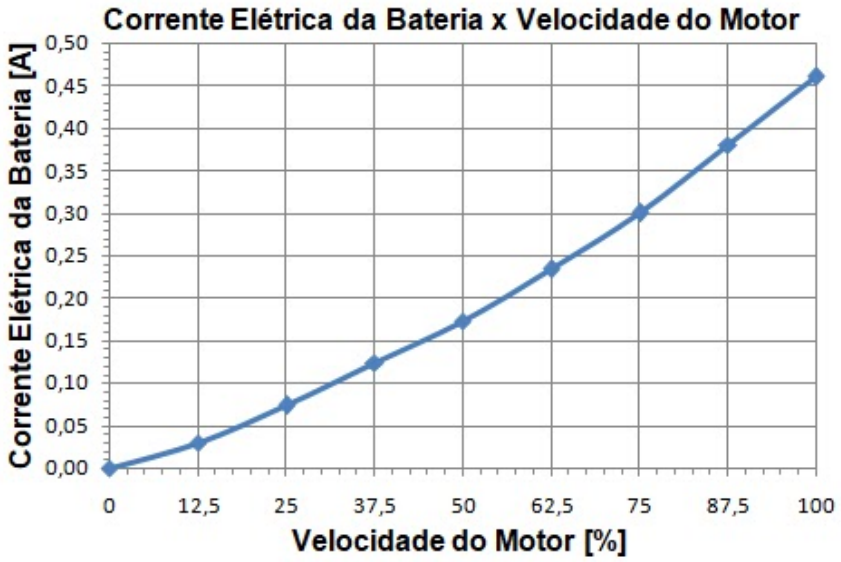
Tabela 4: Valores de corrente elétrica obtidos no experimento

Velocidade do Motor [%]	Corrente Elétrica da Bateria [A]			
	1ª Repetição	2ª Repetição	3ª Repetição	Média
0	0,00	0,00	0,00	0,00
12,5	0,03	0,03	0,03	0,03
25	0,07	0,07	0,08	0,07
37,5	0,13	0,12	0,12	0,12
50	0,17	0,18	0,17	0,17
62,5	0,24	0,23	0,23	0,23
75	0,30	0,29	0,31	0,30
87,5	0,37	0,38	0,39	0,38
100	0,45	0,47	0,46	0,46

Fonte: Autor.

A Figura 20 exibe sob forma de um gráfico o valor médio encontrado para a corrente elétrica da bateria no experimento realizado.

Figura 20: Gráfico da corrente elétrica média obtida no experimento



Fonte: Autor.

A partir das medidas realizadas, é possível identificar que o sistema apresenta um maior consumo de corrente elétrica à medida que a velocidade aplicada ao motor cresce. Na condição de máxima velocidade de operação foi registrado um consumo de corrente elétrica médio de 0,46 A, o que resulta em 18,2 W de potência elétrica drenada da bateria. Em termos percentuais isso equivale a 5,2% da potência nominal do motor (350 w).

4.4. EXPERIMENTOS COM CARGA

Os experimentos desta seção foram denominados “experimentos com carga” pois as medições executadas foram realizadas com o motor *Brushless* DC operando em contato com o solo, onde a massa do veículo elétrico e seu usuário representam a carga mecânica aplicada em seu eixo.

4.4.1. Corrente elétrica em regime permanente

O objetivo deste experimento é avaliar o consumo de corrente elétrica da bateria principal quando o sistema opera em regime permanente com carga mecânica aplicada. A carga aplicada ao motor nesse experimento trata-se da massa do próprio veículo (cerca de 20 kg) somada à massa do usuário (cerca de 80 kg), totalizando aproximadamente 100 kg de massa para ser movimentada.

O protótipo foi testado em um pavilhão comercial com aproximadamente 300 m² de área disponível e piso de concreto com cobertura de borracha em algumas áreas. Esse local foi escolhido pois possui uma faixa retilínea de piso liso com aproximadamente 30 m de comprimento. Devido à limitação de comprimento da faixa de piso liso disponível, o protótipo não pôde ser testado até sua velocidade máxima.

O experimento foi realizado percorrendo uma trajetória em linha reta de aproximadamente 30 m, onde o veículo foi acelerado até quatro diferentes velocidades (2,5 km/h, 5 km/h, 7,5 km/h e 10 km/h). No final do percurso o valor da corrente elétrica observada na bateria principal para cada uma das velocidades foi registrado. Ao total foram executadas três repetições do experimento, de forma que ao final a média aritmética das medidas foi realizada. Durante os testes realizados a tensão elétrica da bateria principal oscilou entre 39,6 V e 39,4 V, dessa forma, considerou-se para esta variável um valor médio de 39,5 V.

A Tabela 5 exibe os dados obtidos nas três repetições do experimento, juntamente com a média aritmética calculada.

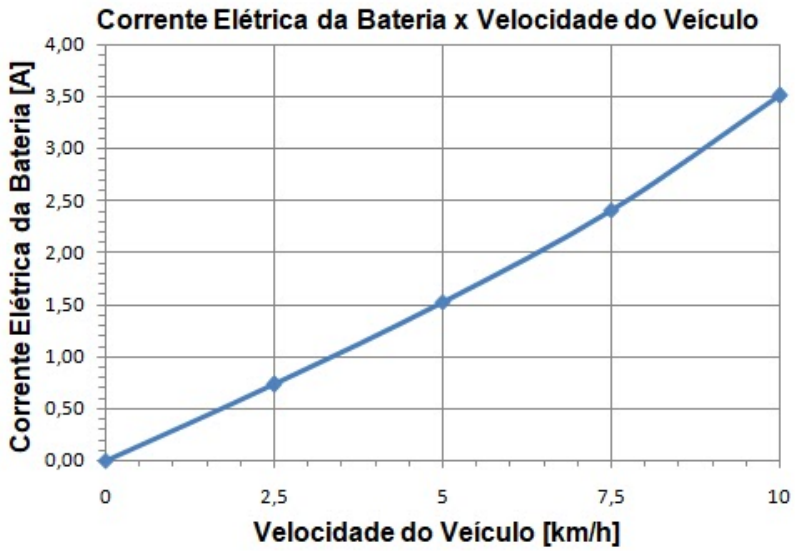
Tabela 5: Valores de corrente elétrica obtidos no experimento

Velocidade do Veículo [km/h]	Corrente Elétrica da Bateria [A]			
	1ª Repetição	2ª Repetição	3ª Repetição	Média
0	0,00	0,00	0,00	0,00
2,5	0,80	0,66	0,72	0,73
5	1,42	1,54	1,61	1,52
7,5	2,38	2,55	2,29	2,41
10	3,64	3,33	3,55	3,51

Fonte: Autor.

A Figura 21 exibe sob forma de um gráfico o valor médio encontrado para a corrente elétrica da bateria no experimento realizado.

Figura 21: Gráfico da corrente elétrica média obtida no experimento



Fonte: Autor.

A partir das medidas realizadas, é possível identificar que o sistema apresenta um maior consumo de corrente elétrica à medida que a velocidade aplicada ao veículo cresce. Na maior velocidade testada (10 km/h) foi registrado um consumo de corrente elétrica médio de 3,51 A, que equivale a aproximadamente 36,1% do limite de corrente nominal do motor, indicando que existe uma grande margem para operação do motor em maiores velocidades ou maiores cargas mecânicas.

4.4.2. Consumo Energético e Autonomia

Para quantificar o consumo energético e a autonomia teórica do protótipo desenvolvido quando operando com carga mecânica aplicada, os valores médios de tensão elétrica e corrente elétrica do experimento anterior foram compilados em uma tabela, juntamente com a velocidade do veículo. Esse conjunto de dados permite identificar a potência elétrica necessária para mover o veículo a cada velocidade, determinando assim seu consumo e autonomia teórica.

O consumo energético por distância percorrida é determinado a partir do quociente da potência elétrica utilizada pela velocidade atingida pelo veículo. Já a autonomia teórica é determinada pelo quociente da capacidade energética da bateria utilizada (633,6 Wh) pelo consumo energético.

A Tabela 6 exibe os valores de consumo energético por distância percorrida e autonomia teórica identificados no sistema para diferentes valores de velocidade do veículo, relacionando ainda os valores médios de tensão, corrente e potência elétrica observados na bateria principal.

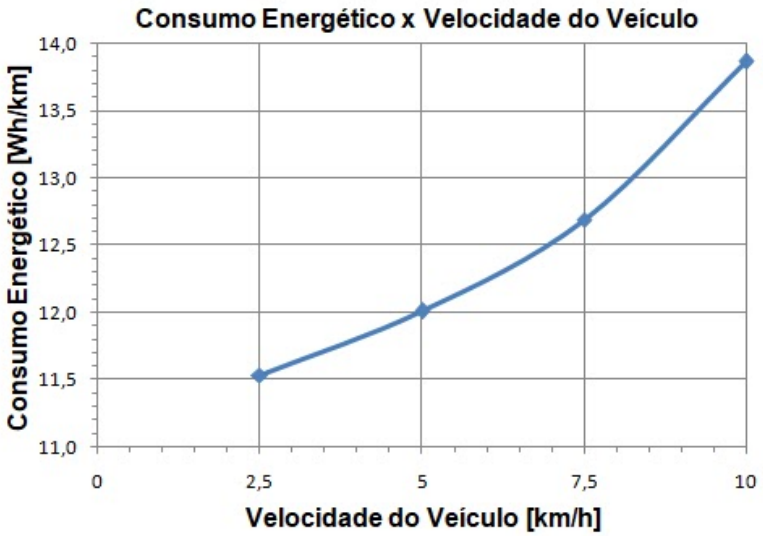
Tabela 6: Valores de tensão, corrente e potência elétrica obtidos

Velocidade do Veículo [km/h]	Valores Médios da Bateria			Consumo Energético [Wh/km]	Autonomia [km]
	Tensão [V]	Corrente [A]	Potência [W]		
0	39,5	0,00	0,0	-	-
2,5	39,5	0,73	28,8	11,5	54,9
5	39,5	1,52	60,0	12,0	52,8
7,5	39,5	2,41	95,2	12,7	49,9
10	39,5	3,51	138,6	13,9	45,7

Fonte: Autor.

A Figura 22 exibe sob forma de um gráfico os valores de consumo energético por distância percorrida identificados para cada velocidade do veículo.

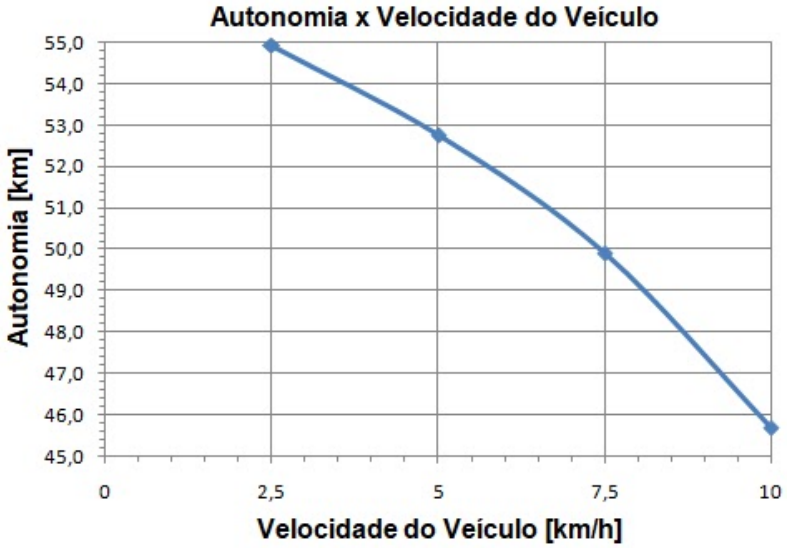
Figura 22: Gráfico do consumo energético obtido no experimento



Fonte: Autor.

A Figura 23 exibe sob forma de um gráfico os valores de autonomia teórica identificados para cada velocidade do veículo.

Figura 23: Gráfico da autonomia estimada



Fonte: Autor.

A partir das medidas realizadas, é possível identificar que o sistema apresenta um maior consumo energético à medida que a velocidade aplicada ao veículo cresce. Observa-se que tal consumo energético apresenta uma relação não-linear com a velocidade, crescendo de forma exponencial no intervalo testado. Já a autonomia teórica também se comporta de forma não-linear, porém de maneira oposta, sendo maior para baixas velocidades.

5. CONCLUSÃO

A arquitetura utilizada e o sistema digital desenvolvido neste trabalho atingiram o objetivo proposto de controlar o sentido de rotação e velocidade de um motor *Brushless* DC através de um dispositivo FPGA. O protótipo de veículo elétrico elaborado permitiu a identificação do comportamento do sistema em uma aplicação real e possibilitou sua parametrização de forma satisfatória, comprovando assim, que o controlador desenvolvido neste trabalho pode ser utilizado em veículos elétricos.

A utilização de um dispositivo FPGA em conjunto com o sistema digital desenvolvido se mostrou uma solução bastante flexível para o controle de motores *Brushless* DC, podendo ser customizada para cada aplicação de forma a atender os requisitos necessários. Essa flexibilidade, devido à natureza reconfigurável do dispositivo FPGA, é o que torna essa solução vantajosa, demonstrando ser uma importante alternativa no controle de motores *Brushless* DC.

O processo de síntese do sistema digital desenvolvido permitiu identificar que são necessários no mínimo 1500 elementos lógicos disponíveis no dispositivo FPGA para que o sistema seja implementado. Esse baixo consumo de elementos lógicos permite que dispositivos FPGA de baixa capacidade sejam utilizados, reduzindo assim o custo do sistema.

O método de comutação trapezoidal 120º se mostrou uma forma bastante simples e eficiente de realizar o acionamento do motor *Brushless* DC. O uso de Sensores Hall para a detecção da posição do rotor permitiu uma fácil integração entre motor e FPGA devido à natureza binária dos sinais destes sensores. Outra vantagem do seu uso é a capacidade de detecção da posição do rotor independentemente do valor de velocidade (inclusive com o rotor parado), proporcionando assim um torque de acionamento consistente mesmo sob baixas velocidades, ideal para aplicações veiculares.

Os testes realizados com o protótipo de veículo elétrico elaborado indicaram que um controlador com comportamento assimétrico apresenta melhores resultados para aplicações veiculares, pois normalmente deseja-se uma aceleração suave do veículo, assim como a possibilidade de realizar frenagens rápidas no caso de uma emergência.

Os experimentos realizados sem a aplicação de carga mecânica ao motor revelaram que o sistema desenvolvido consegue controlar a velocidade de operação do motor com erro nulo em regime permanente para ambos os sentidos de rotação. Nesses experimentos identificou-se ainda que o motor *Brushless* DC utilizado apresenta um consumo de 5,2% de sua potência nominal para atingir a condição de máxima velocidade quando operando à vazio.

Já os experimentos realizados com carga mecânica aplicada ao motor (conjunto veículo-usuário com massa aproximada de 100 kg) demonstraram um aumento do consumo energético por distância percorrida à medida que a velocidade do veículo aumenta. Para uma velocidade de 2,5 km/h observa-se um consumo energético de 11,5 Wh/km, enquanto que, para uma velocidade de 10 km/h observa-se um consumo energético de 13,9 Wh/km.

Com relação à autonomia do veículo, observa-se um comportamento contrário ao identificado para o consumo energético, de forma que, à medida que a velocidade aumenta, a autonomia do veículo se reduz. Para uma velocidade de 2,5 km/h observa-se uma autonomia de 54,9 km, enquanto que, para uma velocidade de 10 km/h observa-se uma autonomia de 45,7 km.

Dessa forma, conclui-se que o protótipo de veículo elétrico testado é mais eficiente em baixas velocidades, pois ele é teoricamente capaz de percorrer 54,9 km a uma velocidade de 2,5 km/h, contra 45,7 km a uma velocidade de 10 km/h.

REFERÊNCIAS

WHAT IS ENERGY? SOURCES OF ENERGY. **Energy Information Administration**, 2019. Disponível em: <https://www.eia.gov/energyexplained/what-is-energy/sources-of-energy.php>. Acesso em: 09 abr. 2020.

OIL: CRUDE AND PETROLEUM PRODUCTS EXPLAINED. **Energy Information Administration**, 2019. Disponível em: <https://www.eia.gov/energyexplained/oil-and-petroleum-products/use-of-oil.php>. Acesso em: 09 abr. 2020.

SORRELL, S. et al. Global Oil Depletion: A Review of the Evidence. **Energy Policy**. V. 38. Pg: 5290-5295. Set. 2010. Disponível em: < https://www.sciencedirect.com/science/article/abs/pii/S0301421510003204>. Acesso em 09 abr. 2020.

KHALIGH, A.; LI, Z. Battery, Ultracapacitor, Fuel Cell, and Hybrid Energy Storage Systems for Electric, Hybrid Electric, Fuel Cell, and Plug-In Hybrid Electric Vehicles: State of the Art. **IEEE Transactions on Vehicular Technology**. V. 59, n. 6, Pg: 2806-2814, Jul. 2010. Disponível em:< https://ieeexplore.ieee.org/document/5446335>. Acesso em 10 abr. 2020.

ALL-ELECTRIC VEHICLES. **U.S. Department of Energy**, 2011. Disponível em: <https://www.fueleconomy.gov/feg/evtech.shtml>. Acesso em 10 abr. 2020.

GIERAS, J. F.; WING, M. **Permanent Magnet Motor Technology: design and applications.** New York: Marcel Dekker, Inc,. p. 590. 2002.

GONELLA, M. C. **Acionamento e Controle Sensorless para Motores Brushless DC Aplicados a Compressores Herméticos para Refrigeração Doméstica.** Dissertação - Escola de Engenharia de São Carlos, Universidade de São Paulo. São Carlos, p. 115. 2006.

THOMPSON, M. AC motor control: DSP, MCU or FPGA?. **EE Times-India**, p. 3. 2009. Disponível em: <https://pt.scribd.com/document/44709040/AC-Motor-Control-DSP-MCU-or-FPGA>. Acesso em: 10 abr. 2020.

FPGAS ENABLE ENERGY-EFFICIENT MOTOR CONTROL IN NEXT-GENERATION SMART HOME APPLIANCES. **ALTERA CORPORATION**, p. 5. Nov. 2008. Disponível em: <https://www.intel.com.br/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01084-fpga-energy-efficient-home-appliance-motor-control.pdf>. Acesso em: 10 abr. 2020.

CARROS ELÉTRICOS. **FGV Energia**, Rio de Janeiro, 2017. Disponível em: <https://fgvenergia.fgv.br/sites/fgvenergia.fgv.br/files/caderno_carros_eletricos-fgv-book.pdf>. Acesso em: 13 abr. 2020.

XIAO, B. et al. Enhanced Regenerative Braking Strategies for Electric Vehicles: Dynamic Performance and Potential Analysis. **Energies,** Nov. 2017. Disponível em:< https://www.mdpi.com/1996-1073/10/11/1875/htm>. Acesso em 13 abr. 2020.

MONTEIRO, J. R. B. A. **TRANSFORMAÇÃO DQ NÃO SENOIDAL PARA MÁQUINAS SÍNCRONAS COM ÍMÃ PERMANENTE NO ROTOR**. Tese – Escola de Engenharia de São Carlos, Universidade de São Paulo. São Carlos, p. 108. 2002.

MILLER, T. J. E. **Brushless Permanent – Magnet and Relutance Motor Drives.** Oxford: Claredon Press. p. 207. 1989.

JULIANI, A. D. P. **Análise do Campo Magnético de um Motor de Ímã Permanente no Rotor Utilizando o Método dos Elementos Finitos.** Dissertação - Escola de Engenharia de São Carlos, Universidade de São Paulo. São Carlos, p. 116. 2007.

AKIN, B.; BHARDWAJ, M. Trapezoidal Control of BLDC Motors Using Hall Effect Sensors. **Texas Instruments**, 2011. Disponível em: <http://www.ti.com /lit/an/sprabz4/sprabz4.pdf>. Acesso em: 14 maio 2020.

AKIN, B.; BHARDWAJ, M. Sensorless Trapezoidal Control of BLDC Motors. **Texas Instruments**, 2015. Disponível em: <http://www.ti.com/lit/an/sprabq7a/sprabq7a.pdf?%26ts%3D1589820791819&sa=D&ust=1589829819068000&usg=AFQjCNHz0s02I_qOCyocjJFvLig3aS2Q5g>. Acesso em: 14 maio 2020.

HENDERSHOT, J. R.; MILLER, T. J. E. **Design of Brushless Permanent-Magnet Motors.** Hillsboro: Magna Physics Pub. p. 584. 1994

RANDRIAKOTONJANAHARY, T. S. A low-cost, small scale Unmanned Aerial Vehicle capable of a real-time onboard deep learning-based object detection system. **ResearchGate,** 2019. Disponível em:<https://www.researchgate.net/publication/338954801_A_low-cost_small_scale_Unmanned_Aerial_Vehicle_capable_of_a_real-time_onboard_deep_learning-based_object_detection_system>. Acesso em 13 abr. 2020.

YE, J.; HUANG Z. Design and Realization of Control System of Brushless DC Motor Based on ARM. **Journal of Physics**: Conference Series 1087. Set. 2018. Disponível em: <https://www.researchgate.net/publication/328036900_Design _and_ Realization_of_Control_System_of_Brushless_DC_Motor_Based_on_ARM>. Acesso em 13 abr. 2020.

WHAT IS AN FPGA?. **XILINX, 2021**. Disponível em: <https://www.xilinx.com /products/silicon-devices/fpga/what-is-an-fpga.html>. Acesso em: 22 abr. 2021.

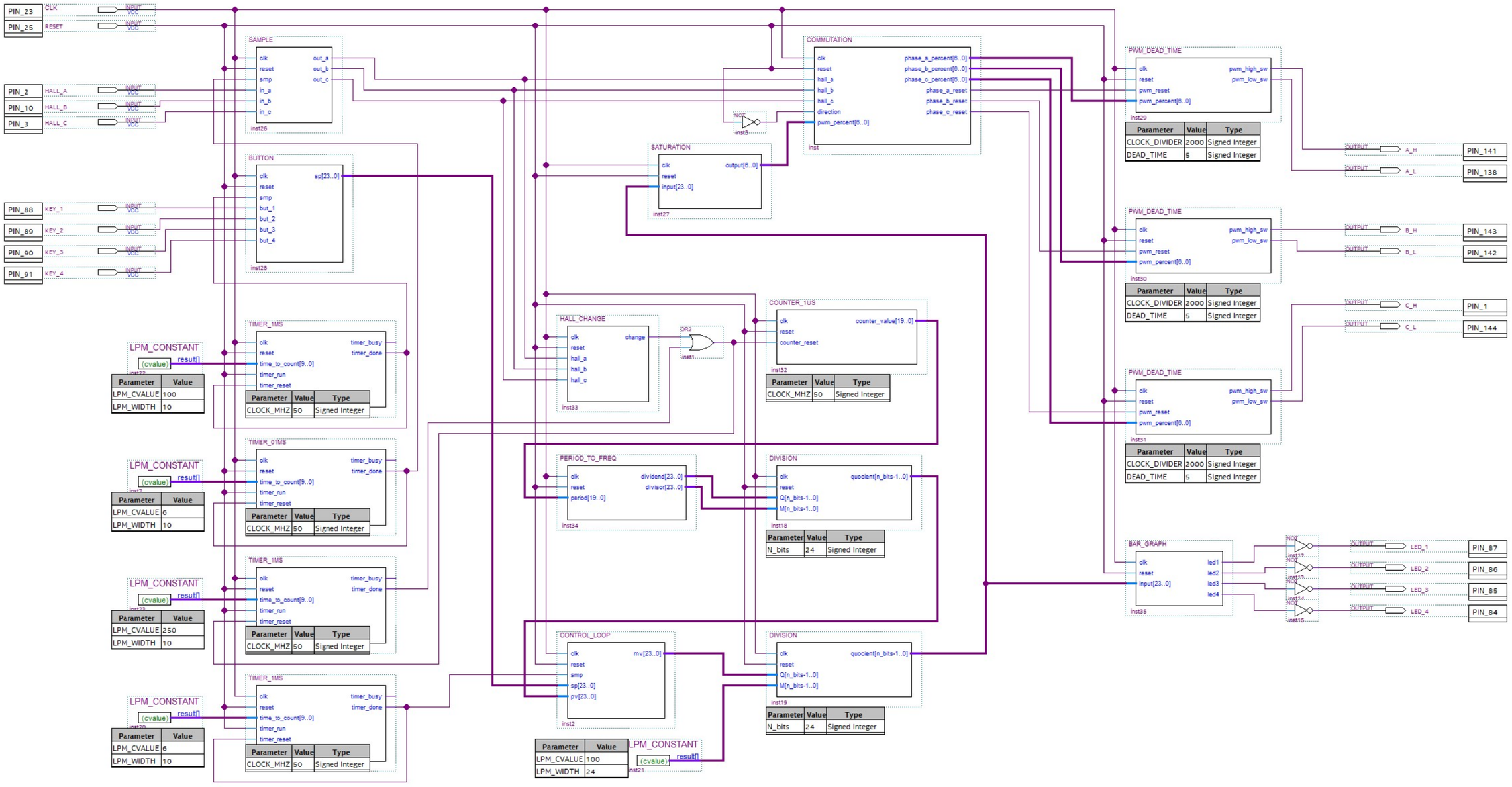
FPGA FUNDAMENTALS. **National Instruments**, 2020. Disponível em: <https://www.ni.com/pt-br/innovations/white-papers/08/fpga-fundamentals.html>. Acesso em: 22 abr. 2021.

OGATA, K. **ENGENHARIA DE CONTROLE MODERNO.** São Paulo: Pearson Prentice Hall 5ª. ed. p.815. 2010.

RUI ZHI YAN FA. **RZRD**, c2015. Página inicial. Disponível em: <http://www.rzrd.net/>. Acesso em: 22 abr. 2021.

NON-RESTORING DIVISION FOR UNSIGNED INTEGER. **Tutorialspoint**, 2018. Disponível em: <https://tutorialspoint.dev/computer-science/computer-organization-and-architecture /non-restoring-division-unsigned-integer>. Acesso em: 9 mar. 2021.

ANEXO 1 - DIAGRAMA DO SISTEMA DIGITAL DESENVOLVIDO



Parameter	Value	Type
CLOCK_DIVIDER	2000	Signed Integer
DEAD_TIME	5	Signed Integer

Parameter	Value
LPM_CVALUE	100
LPM_WIDTH	10

Parameter	Value
LPM_CVALUE	6
LPM_WIDTH	10

Parameter	Value
LPM_CVALUE	250
LPM_WIDTH	10

Parameter	Value
LPM_CVALUE	6
LPM_WIDTH	10

Parameter	Value
LPM_CVALUE	100
LPM_WIDTH	24

Parameter	Value	Type
CLOCK_MHZ	50	Signed Integer

Parameter	Value	Type
N_bits	24	Signed Integer

ANEXO 2 - CÓDIGOS VHDL DESENVOLVIDOS

SAMPLE.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY SAMPLE IS
PORT
(
   clk                  : IN    STD_LOGIC;
   reset          : IN    STD_LOGIC;
   smp                  : IN    STD_LOGIC;
   in_a                 : IN    STD_LOGIC;
   in_b           : IN    STD_LOGIC;
   in_c                 : IN    STD_LOGIC;
   out_a          : OUT   STD_LOGIC;
   out_b          : OUT   STD_LOGIC;
   out_c          : OUT   STD_LOGIC
);
END ENTITY;
ARCHITECTURE behavioral OF SAMPLE IS
   SIGNAL smp_reg                  : STD_LOGIC;
   SIGNAL out_a_reg          : STD_LOGIC;
   SIGNAL out_b_reg          : STD_LOGIC;
   SIGNAL out_c_reg          : STD_LOGIC;
BEGIN
   PROCESS (clk,reset)
   BEGIN
         IF (clk'EVENT AND clk = '1') THEN
               IF (reset = '0') THEN
                     out_a <= '0';
                     out_b <= '0';
                     out_c <= '0';
                     smp_reg <= '0';
                     out_a_reg <= '0';
                     out_b_reg <= '0';
                     out_c_reg <= '0';
               ELSE
                     smp_reg <= smp;
                     out_a <= out_a_reg;
                     out_b <= out_b_reg;
                     out_c <= out_c_reg;
                     IF (smp_reg = '0') AND (smp = '1') THEN
                           out_a_reg <= in_a;
                           out_b_reg <= in_b;
                           out_c_reg <= in_c;
                     ELSE
                           out_a_reg <= out_a_reg;
                           out_b_reg <= out_b_reg;
                           out_c_reg <= out_c_reg;
                     END IF;
               END IF;
         END IF;
   END PROCESS;
END ARCHITECTURE;
```

BUTTON.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY BUTTON IS
PORT
(
   clk                  : IN    STD_LOGIC;
   reset                : IN  STD_LOGIC;
   smp                  : IN    STD_LOGIC;
   but_1                : IN  STD_LOGIC;
   but_2                : IN  STD_LOGIC;
   but_3                : IN  STD_LOGIC;
   but_4                : IN  STD_LOGIC;
   sp                           : OUT STD_LOGIC_VECTOR (23 downto 0)
);
END ENTITY;
ARCHITECTURE behavioral OF BUTTON IS
   SIGNAL smp_reg               : STD_LOGIC;
   SIGNAL but_1_reg         : STD_LOGIC;
   SIGNAL but_11_reg        : STD_LOGIC;
   SIGNAL but_2_reg         : STD_LOGIC;
   SIGNAL but_22_reg        : STD_LOGIC;
   SIGNAL but_3_reg         : STD_LOGIC;
   SIGNAL but_33_reg        : STD_LOGIC;
   SIGNAL but_4_reg         : STD_LOGIC;
   SIGNAL but_44_reg        : STD_LOGIC;
   SIGNAL sp_reg                : INTEGER range 0 to 16777215;
   SIGNAL sp_reg_calc    : INTEGER range 0 to 16777215;
BEGIN
   PROCESS (clk,reset)
   BEGIN
         IF (clk'EVENT AND clk = '1') THEN
               IF (reset = '0') THEN
                     sp <= (others => '0');
                     smp_reg <= '1';
                     but_1_reg <= '1';
                     but_11_reg <= '1';
                     but_2_reg <= '1';
                     but_22_reg <= '1';
                     but_3_reg <= '1';
                     but_33_reg <= '1';
                     but_4_reg <= '1';
                     but_44_reg <= '1';
                     sp_reg <= 0;
               ELSE
                     smp_reg <= smp;
                     sp <= std_logic_vector(to_unsigned(sp_reg, sp'length));
                     IF (smp_reg = '0') AND (smp = '1') THEN
                           but_1_reg <= but_1;
                           but_11_reg <= but_1_reg;
                           but_2_reg <= but_2;
                           but_22_reg <= but_2_reg;
                           but_3_reg <= but_3;
                           but_33_reg <= but_3_reg;
                           but_4_reg <= but_4;
                           but_44_reg <= but_4_reg;
                     ELSE
                           but_1_reg <= but_1_reg;
                           but_11_reg <= but_11_reg;
                           but_2_reg <= but_2_reg;
                           but_22_reg <= but_22_reg;
                           but_3_reg <= but_3_reg;
                           but_33_reg <= but_33_reg;
                           but_4_reg <= but_4_reg;
                           but_44_reg <= but_44_reg;
                     END IF;
                     IF (sp_reg_calc > 1600) THEN
                           sp_reg <= 1600;
                     ELSE
                           sp_reg <= sp_reg_calc;
                     END IF;
               END IF;
         END IF;
```

```
END PROCESS;
PROCESS (clk,reset)
BEGIN
        IF (clk'EVENT AND clk = '1') THEN
                IF (reset = '0') THEN
                        sp_reg_calc <= 0;
                ELSE
                        IF (smp_reg = '0') AND (smp = '1') THEN
                                IF (but_11_reg = '1') AND (but_1_reg = '0') THEN
                                        sp_reg_calc <= 0;
                                ELSE
                                        IF (but_22_reg = '1') AND (but_2_reg = '0') THEN
                                                IF (sp_reg_calc > 199) THEN
                                                        sp_reg_calc <= sp_reg_calc - 200;
                                                ELSE
                                                        sp_reg_calc <= sp_reg_calc;
                                                END IF;
                                        ELSE
                                                IF (but_33_reg = '1') AND (but_3_reg = '0') THEN
                                                 IF (sp_reg_calc < 1401) THEN
                                                         sp_reg_calc <= sp_reg_calc + 200;
                                                 ELSE
                                                        sp_reg_calc <= sp_reg_calc;
                                                 END IF;
                                                ELSE
                                                 IF (but_44_reg = '1') AND (but_4_reg = '0') THEN
                                                        sp_reg_calc <= 0;
                                                 ELSE
                                                        sp_reg_calc <= sp_reg_calc;
                                                 END IF;
                                                END IF;
                                        END IF;
                                END IF;
                        ELSE
                                sp_reg_calc <= sp_reg_calc;
                        END IF;
                END IF;
        END IF;
  END PROCESS;
END ARCHITECTURE;
```

TIMER_1MS.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY TIMER_1MS IS
GENERIC
(
   CLOCK_MHZ : INTEGER range 0 to 255 := 50
);
PORT
(
   clk                  : IN    STD_LOGIC;
   reset      : IN  STD_LOGIC;
   time_to_count   : IN  STD_LOGIC_VECTOR (9 downto 0);
   timer_run            : IN  STD_LOGIC;
   timer_reset          : IN  STD_LOGIC;
   timer_busy           : OUT STD_LOGIC;
   timer_done           : OUT STD_LOGIC
);
END ENTITY;
ARCHITECTURE behavioral OF TIMER_1MS IS
   SIGNAL   timer_busy_reg              : STD_LOGIC;
   SIGNAL   timer_done_reg              : STD_LOGIC;
   SIGNAL   sub_timer_value    : INTEGER range 0 to 65535;
   SIGNAL   timer_value                 : INTEGER range 0 to 1023;
BEGIN
   PROCESS (clk,reset)
   BEGIN
          IF (clk'EVENT AND clk = '1') THEN
                 IF (reset = '0') THEN
                        timer_busy <= '0';
                        timer_done <= '0';
                        timer_busy_reg <= '0';
                        timer_done_reg <= '0';
                        sub_timer_value <= 0;
                        timer_value <= 0;
                 ELSE
                        timer_busy <= timer_busy_reg;
                        timer_done <= timer_done_reg;
                        IF timer_reset = '1' THEN
                               timer_busy_reg <= '0';
                               timer_done_reg <= '0';
                               sub_timer_value <= 0;
                               timer_value <= 0;
                        ELSE
                               IF timer_run = '1' THEN
                                   timer_busy_reg <= '1';
                                      IF (timer_value < to_integer(unsigned(time_to_count))) THEN
                                           IF (sub_timer_value < ((CLOCK_MHZ * 1000) - 1)) THEN
                                                  sub_timer_value <= sub_timer_value + 1;
                                           ELSE
                                                  sub_timer_value <= 0;
                                                  timer_value <= timer_value + 1;
                                           END IF;
                                      ELSE
                                           timer_done_reg <= '1';
                                      END IF;
                               ELSE
                                          timer_value <= timer_value;
                               END IF;
                        END IF;
                 END IF;
          END IF;
   END PROCESS;
END ARCHITECTURE;
```

TIMER_01MS.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY TIMER_01MS IS
GENERIC
(
   CLOCK_MHZ : INTEGER range 0 to 255 := 50
);
PORT
(
   clk                  : IN    STD_LOGIC;
   reset                : IN  STD_LOGIC;
   time_to_count    : IN  STD_LOGIC_VECTOR (9 downto 0);
   timer_run        : IN  STD_LOGIC;
   timer_reset      : IN  STD_LOGIC;
   timer_busy       : OUT STD_LOGIC;
   timer_done       : OUT STD_LOGIC
);
END ENTITY;
ARCHITECTURE behavioral OF TIMER_01MS IS
   SIGNAL   timer_busy_reg          : STD_LOGIC;
   SIGNAL   timer_done_reg          : STD_LOGIC;
   SIGNAL   sub_timer_value    : INTEGER range 0 to 65535;
   SIGNAL   timer_value             : INTEGER range 0 to 1023;
BEGIN
   PROCESS (clk,reset)
   BEGIN
         IF (clk'EVENT AND clk = '1') THEN
               IF (reset = '0') THEN
                     timer_busy <= '0';
                     timer_done <= '0';
                     timer_busy_reg <= '0';
                     timer_done_reg <= '0';
                     sub_timer_value <= 0;
                     timer_value <= 0;
               ELSE
                     timer_busy <= timer_busy_reg;
                     timer_done <= timer_done_reg;
                     IF timer_reset = '1' THEN
                           timer_busy_reg <= '0';
                           timer_done_reg <= '0';
                           sub_timer_value <= 0;
                           timer_value <= 0;
                     ELSE
                           IF timer_run = '1' THEN
                              timer_busy_reg <= '1';
                              IF (timer_value < to_integer(unsigned(time_to_count))) THEN
                                 IF (sub_timer_value < ((CLOCK_MHZ * 100) - 1)) THEN
                                       sub_timer_value <= sub_timer_value + 1;
                                 ELSE
                                       sub_timer_value <= 0;
                                       timer_value <= timer_value + 1;
                                 END IF;
                              ELSE
                                    timer_done_reg <= '1';
                              END IF;
                           ELSE
                                 timer_value <= timer_value;
                           END IF;
                     END IF;
               END IF;
         END IF;
   END PROCESS;
END ARCHITECTURE;
```

HALL_CHANGE.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY HALL_CHANGE IS
PORT
(
   clk                              : IN    STD_LOGIC;
   reset                    : IN  STD_LOGIC;
   hall_a                           : IN    STD_LOGIC;
   hall_b                           : IN    STD_LOGIC;
   hall_c                           : IN    STD_LOGIC;
   change                           : OUT STD_LOGIC
);
END ENTITY;
ARCHITECTURE behavioral OF HALL_CHANGE IS
   SIGNAL hall_a_reg        : STD_LOGIC;
   SIGNAL hall_b_reg        : STD_LOGIC;
   SIGNAL hall_c_reg        : STD_LOGIC;
BEGIN
   PROCESS (clk,reset)
   BEGIN
          IF (clk'EVENT AND clk = '1') THEN
                 IF (reset = '0') THEN
                        change <= '0';
                        hall_a_reg <= '0';
                        hall_b_reg <= '0';
                        hall_c_reg <= '0';
                 ELSE
                        hall_a_reg <= hall_a;
                        hall_b_reg <= hall_b;
                        hall_c_reg <= hall_c;
                        IF (hall_a_reg = hall_a) THEN
                               change <= '0';
                        ELSE
                               change <= '1';
                        END IF;
                 END IF;
          END IF;
   END PROCESS;
END ARCHITECTURE;
```

COUNTER_1US.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY COUNTER_1US IS
GENERIC
(
   CLOCK_MHZ : INTEGER range 0 to 255 := 50
);
PORT
(
   clk                  : IN    STD_LOGIC;
   reset                : IN  STD_LOGIC;
   counter_reset    : IN  STD_LOGIC;
   counter_value    : OUT STD_LOGIC_VECTOR (19 downto 0)
);
END ENTITY;
ARCHITECTURE behavioral OF COUNTER_1US IS
   SIGNAL   sub_counter_value_reg    : INTEGER range 0 to 255;
   SIGNAL   counter_value_reg                : INTEGER range 0 to 1048575;
   SIGNAL   counter_value_reg_1        : INTEGER range 0 to 1048575;
   SIGNAL   counter_value_reg_2        : INTEGER range 0 to 1048575;
   SIGNAL   counter_value_reg_3        : INTEGER range 0 to 1048575;
   SIGNAL   counter_value_reg_4        : INTEGER range 0 to 1048575;
   SIGNAL   counter_reset_reg                : STD_LOGIC;
BEGIN
   PROCESS (clk,reset)
   BEGIN
         IF (clk'EVENT AND clk = '1') THEN
               IF (reset = '0') THEN
                     sub_counter_value_reg <= 0;
                     counter_value_reg <= 0;
                     counter_value_reg_1 <= 0;
                     counter_value_reg_2 <= 0;
                     counter_value_reg_3 <= 0;
                     counter_value_reg_4 <= 0;
                     counter_reset_reg <= '0';
               ELSE
                     counter_reset_reg <= counter_reset;
                        IF (counter_reset_reg = '0') AND (counter_reset = '0')  THEN
                              IF (sub_counter_value_reg < (CLOCK_MHZ - 1)) THEN
                                 sub_counter_value_reg <= sub_counter_value_reg + 1;
                              ELSE
                                 sub_counter_value_reg <= 0;
                                 IF (counter_value_reg < 249000) THEN
                                       counter_value_reg <= counter_value_reg + 1;
                                 ELSE
                                          counter_value_reg <= counter_value_reg;
                                 END IF;
                              END IF;
                        END IF;
                        IF (counter_reset_reg = '0') AND (counter_reset = '1')  THEN
                              counter_value_reg_1 <= counter_value_reg;
                              counter_value_reg_2 <= counter_value_reg_1;
                              counter_value_reg_3 <= counter_value_reg_2;
                              counter_value_reg_4 <= counter_value_reg_3;
                        END IF;
                        IF (counter_reset_reg = '1') AND (counter_reset = '1')  THEN
                              sub_counter_value_reg <= 0;
                              counter_value_reg <= 0;
                        END IF;
                        IF (counter_reset_reg = '1') AND (counter_reset = '0')  THEN
                              sub_counter_value_reg <= 0;
                              counter_value_reg <= 0;
                        END IF;
               END IF;
         END IF;
   END PROCESS;

   PROCESS (clk)
   BEGIN
         IF (clk'EVENT AND clk = '1') THEN
                  counter_value    <=    std_logic_vector(to_unsigned(((counter_value_reg_1    +    counter_value_reg_2    +
counter_value_reg_3 + counter_value_reg_4) / 4), counter_value'length));
```

```
        END IF;
    END PROCESS;
END ARCHITECTURE;
```

PERIOD_TO_FREQ.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY PERIOD_TO_FREQ IS
PORT
(
   clk                  : IN    STD_LOGIC;
   reset     : IN    STD_LOGIC;
   period               : IN  STD_LOGIC_VECTOR (19 downto 0);
   dividend             : OUT STD_LOGIC_VECTOR (23 downto 0);
   divisor              : OUT STD_LOGIC_VECTOR (23 downto 0)
);
END ENTITY;
ARCHITECTURE behavioral OF PERIOD_TO_FREQ IS
   SIGNAL period_reg                    : INTEGER range 0 to 1048575;
BEGIN
   PROCESS (clk)
   BEGIN
            IF (clk'EVENT AND clk = '1') THEN
                     IF (reset = '0') THEN
                              dividend <= std_logic_vector(to_unsigned(0, dividend'length));
                              divisor <= std_logic_vector(to_unsigned(1, divisor'length));
                              period_reg <= 0;
                     ELSE
                              period_reg <= to_integer(unsigned(period));
                              IF (period_reg > 2940) THEN
                                 IF (period_reg > 248000) THEN
                               dividend <= std_logic_vector(to_unsigned(0, dividend'length));
                                   divisor <= std_logic_vector(to_unsigned(1, divisor'length));
                              ELSE
                                   dividend <= std_logic_vector(to_unsigned(10000000, dividend'length));
                             divisor <= std_logic_vector(to_unsigned((period_reg * 2), divisor'length));
                            END IF;
                              ELSE
                                dividend <= std_logic_vector(to_unsigned(10000000, dividend'length));
                             divisor <= std_logic_vector(to_unsigned(5880, divisor'length));
                              END IF;
                     END IF;
            END IF;
   END PROCESS;
END ARCHITECTURE;
```

DIVISION.vhd

```
--Non Restoring Division Algorithm -> (Q/M)
-- N = num of bits of Q
-- M = divisor
-- A = acumulator
-- Q = dividend

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY DIVISION IS
GENERIC
(
   N_bits              : integer := 24
);
PORT
(
   clk                        : IN  STD_LOGIC;
   reset              : IN  STD_LOGIC;
   Q                          : IN  STD_LOGIC_VECTOR (N_bits - 1 downto 0);
   M                          : IN  STD_LOGIC_VECTOR (N_bits - 1 downto 0);
   quocient                   : OUT STD_LOGIC_VECTOR (N_bits - 1 downto 0)
);
END ENTITY;
ARCHITECTURE behavioral OF DIVISION IS
   TYPE               STATE_TYPE                  IS (state_0, state_2, state_3, state_4, state_5, state_6, state_7, state_8,
state_9, state_10, state_11, state_12, state_13, state_14, state_15);

   SIGNAL    state                                : STATE_TYPE;
   SIGNAL    M_reg                                : UNSIGNED (N_bits downto 0);
   SIGNAL    N_reg                                : INTEGER range 0 to N_bits;
   SIGNAL    AQ_reg                               : UNSIGNED (N_bits * 2 downto 0);
BEGIN
   PROCESS (clk,reset)
   BEGIN
          IF (clk'EVENT AND clk = '1') THEN
                 IF (reset = '0') THEN
                        quocient <= (others=>'0');
                        state <= state_0;
                        M_reg <= (others=>'0');
                        N_reg <= 0;
                        AQ_reg <= (others=>'0');
                 ELSE
                        CASE state IS
                               WHEN state_0 =>
                                      state <= state_2;
                               WHEN state_2 =>
                                      N_reg <= N_bits;
                                      M_reg (N_bits - 1 downto 0) <= UNSIGNED(M);
                                      AQ_reg <= (to_unsigned(0, (N_bits + 1))) & (UNSIGNED(Q));
                                      state <= state_3;
                               WHEN state_3 =>
                                      IF (AQ_reg(N_bits * 2) = '1') THEN
                                             state <= state_4;
                                      ELSE
                                             state <= state_6;
                                      END IF;
                               WHEN state_4 =>
                                      AQ_reg <= shift_left(unsigned(AQ_reg), 1);
                                      state <= state_5;
                               WHEN state_5 =>
                                      AQ_reg (N_bits*2 downto N_bits) <= (AQ_reg (N_bits*2 downto N_bits) +
M_reg);
                                      state <= state_8;
                               WHEN state_6 =>
                                      AQ_reg <= shift_left(unsigned(AQ_reg), 1);
                                      state <= state_7;
                               WHEN state_7 =>
                                      AQ_reg (N_bits*2 downto N_bits) <= (AQ_reg (N_bits*2 downto N_bits) -
M_reg);
                                      state <= state_8;
                               WHEN state_8 =>
                                      IF (AQ_reg(N_bits * 2) = '1') THEN
                                             state <= state_9;
```

```
                                        ELSE
                                                state <= state_10;
                                        END IF;
                                WHEN state_9 =>
                                        AQ_reg(0) <= '0';
                                        state <= state_11;
                                WHEN state_10 =>
                                        AQ_reg(0) <= '1';
                                        state <= state_11;
                                WHEN state_11 =>
                                        N_reg <= N_reg - 1;
                                        state <= state_12;
                                WHEN state_12 =>
                                        IF (N_reg = 0) THEN
                                                state <= state_13;
                                        ELSE
                                                state <= state_3;
                                        END IF;
                                WHEN state_13 =>
                                        IF (AQ_reg(N_bits * 2) = '1') THEN
                                                state <= state_14;
                                        ELSE
                                                state <= state_15;
                                        END IF;
                                WHEN state_14 =>
                                        AQ_reg (N_bits*2 downto N_bits) <= (AQ_reg (N_bits*2 downto N_bits) +
M_reg);
                                        state <= state_15;
                                WHEN state_15 =>
                                        quocient <= std_logic_vector(AQ_reg (N_bits - 1 downto 0));
                                        state <= state_2;
                                WHEN OTHERS =>
                                        state <= state_0;
                        END CASE;
                END IF;
        END IF;
  END PROCESS;
END ARCHITECTURE;
```

CONTROL_LOOP.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY CONTROL_LOOP IS
PORT
(
   clk                  : IN    STD_LOGIC;
   reset          : IN  STD_LOGIC;
   smp                  : IN  STD_LOGIC;
   sp                   : IN  STD_LOGIC_VECTOR (23 downto 0);
   pv                   : IN  STD_LOGIC_VECTOR (23 downto 0);
   mv                   : OUT STD_LOGIC_VECTOR (23 downto 0)
);
END ENTITY;
ARCHITECTURE behavioral OF CONTROL_LOOP IS
   SIGNAL smp_reg                : STD_LOGIC;
   SIGNAL sp_reg                 : INTEGER range 0 to 16777215;
   SIGNAL pv_reg                 : INTEGER range 0 to 16777215;
   SIGNAL mv_reg                 : INTEGER range 0 to 16777215;
   SIGNAL E_reg          : INTEGER range -16777216 to 16777215;
   SIGNAL P_reg          : INTEGER range -16777216 to 16777215;
   SIGNAL I_reg          : INTEGER range -16777216 to 16777215;
   SIGNAL mv_calc_reg    : INTEGER range -16777216 to 16777215;
BEGIN
   PROCESS (clk,reset)
   BEGIN
         IF (clk'EVENT AND clk = '1') THEN
               IF (reset = '0') THEN
                     mv <= (others => '0');
                     smp_reg <= '0';
                     sp_reg <= 0;
                     pv_reg <= 0;
                     mv_reg <= 0;
                     E_reg <= 0;
                     P_reg <= 0;
                     I_reg <= 0;
                     mv_calc_reg <= 0;
               ELSE
                     smp_reg <= smp;
                     sp_reg <= to_integer(unsigned(sp));
                     pv_reg <= to_integer(unsigned(pv));
                     mv <= std_logic_vector(to_unsigned(mv_reg, mv'length));
                     E_reg <= (sp_reg - pv_reg);
                     IF (smp_reg = '0') AND (smp = '1') THEN
                           IF (E_reg > 0) THEN
                                 P_reg <= (E_reg * 8);
                                 IF (I_reg > 10000) THEN
                                       I_reg <= 10001;
                                 ELSE
                                       I_reg <= I_reg + (E_reg / 32);
                                 END IF;
                           ELSE
                                 P_reg <= 0;
                                 IF (E_reg < 0) THEN
                                       IF (I_reg < -10000) THEN
                                             I_reg <= -10001;
                                       ELSE
                                             I_reg <= I_reg + (E_reg / 4);
                                       END IF;
                                 ELSE
                                       I_reg <= I_reg;
                                 END IF;
                           END IF;
                     END IF;
                     mv_calc_reg <= (P_reg + I_reg);
                     IF (mv_calc_reg < 0) THEN
                           mv_reg <= 0;
                     ELSE
                           IF (mv_calc_reg > 10000) THEN
                                 mv_reg <= 10000;
                           ELSE
                                 mv_reg <= mv_calc_reg;
                           END IF;
```

```
                    END IF;
                END IF;
            END IF;
        END PROCESS;
    END ARCHITECTURE;
```

SATURATION.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY SATURATION IS
PORT
(
   clk                        : IN    STD_LOGIC;
   reset                      : IN    STD_LOGIC;
   input                      : IN  STD_LOGIC_VECTOR (23 downto 0);
   output                     : OUT STD_LOGIC_VECTOR (6 downto 0)
);
END ENTITY;
ARCHITECTURE behavioral OF SATURATION IS
   SIGNAL input_reg                  : INTEGER range 0 to 16777215;
BEGIN
   PROCESS (clk,reset)
   BEGIN
         IF (clk'EVENT AND clk = '1') THEN
               IF (reset = '0') THEN
                     output <= (others => '0');
                     input_reg <= 0;
               ELSE
                     input_reg <= to_integer(unsigned(input));
                     IF (input_reg > 99) THEN
                           output <= std_logic_vector(to_unsigned(99, output'length));
                     ELSE
                           output <= std_logic_vector(to_unsigned(input_reg, output'length));
                     END IF;
               END IF;
         END IF;
   END PROCESS;
END ARCHITECTURE;
```

COMMUTATION.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY COMMUTATION IS
PORT
(
   clk                              : IN    STD_LOGIC;
   reset                    : IN  STD_LOGIC;
   hall_a                           : IN    STD_LOGIC;
   hall_b                           : IN    STD_LOGIC;
   hall_c                           : IN    STD_LOGIC;
   direction                        : IN    STD_LOGIC;
   pwm_percent              : IN  STD_LOGIC_VECTOR (6 downto 0);
   phase_a_percent                  : OUT  STD_LOGIC_VECTOR (6 downto 0);
   phase_b_percent                  : OUT  STD_LOGIC_VECTOR (6 downto 0);
   phase_c_percent                  : OUT  STD_LOGIC_VECTOR (6 downto 0);
   phase_a_reset            : OUT  STD_LOGIC;
   phase_b_reset            : OUT  STD_LOGIC;
   phase_c_reset            : OUT  STD_LOGIC
);
END ENTITY;
ARCHITECTURE behavioral OF COMMUTATION IS
   SIGNAL hall_a_reg                : STD_LOGIC;
   SIGNAL hall_b_reg                : STD_LOGIC;
   SIGNAL hall_c_reg                : STD_LOGIC;
   SIGNAL pwm_percent_reg           : STD_LOGIC_VECTOR (6 downto 0);
   SIGNAL phase_a_percent_reg   : STD_LOGIC_VECTOR (6 downto 0);
   SIGNAL phase_b_percent_reg   : STD_LOGIC_VECTOR (6 downto 0);
   SIGNAL phase_c_percent_reg   : STD_LOGIC_VECTOR (6 downto 0);
   SIGNAL phase_a_reset_reg     : STD_LOGIC;
   SIGNAL phase_b_reset_reg     : STD_LOGIC;
   SIGNAL phase_c_reset_reg     : STD_LOGIC;
BEGIN
   PROCESS (clk,reset)
   BEGIN
         IF (clk'EVENT AND clk = '1') THEN
               IF (reset = '0') THEN
                     phase_a_percent <= (others => '0');
                     phase_b_percent <= (others => '0');
                     phase_c_percent <= (others => '0');
                     phase_a_reset <= '1';
                     phase_b_reset <= '1';
                     phase_c_reset <= '1';
                     hall_a_reg <= '0';
                     hall_b_reg <= '0';
                     hall_c_reg <= '0';
                     pwm_percent_reg <= (others => '0');
                     phase_a_percent_reg <= (others => '0');
                     phase_b_percent_reg <= (others => '0');
                     phase_c_percent_reg <= (others => '0');
                     phase_a_reset_reg <= '1';
                     phase_b_reset_reg <= '1';
                     phase_c_reset_reg <= '1';
               ELSE
                     hall_a_reg <= hall_a;
                     hall_b_reg <= hall_b;
                     hall_c_reg <= hall_c;
                     pwm_percent_reg <= pwm_percent;
                     phase_a_percent <= phase_a_percent_reg;
                     phase_b_percent <= phase_b_percent_reg;
                     phase_c_percent <= phase_c_percent_reg;
                     phase_a_reset <= phase_a_reset_reg;
                     phase_b_reset <= phase_b_reset_reg;
                     phase_c_reset <= phase_c_reset_reg;
                     IF (hall_a_reg = '1') AND (hall_b_reg = '0') AND (hall_c_reg = '0') THEN
                           IF (direction = '0') THEN
                                 phase_a_percent_reg <= pwm_percent_reg;
                                 phase_b_percent_reg <= (others => '0');
                                 phase_c_percent_reg <= (others => '0');
                                 phase_a_reset_reg <= '0';
                                 phase_b_reset_reg <= '0';
                                 phase_c_reset_reg <= '1';
                           ELSE
```

```
                phase_a_percent_reg <= (others => '0');
                phase_b_percent_reg <= pwm_percent_reg;
                phase_c_percent_reg <= (others => '0');
                phase_a_reset_reg <= '0';
                phase_b_reset_reg <= '0';
                phase_c_reset_reg <= '1';
        END IF;
END IF;
IF (hall_a_reg = '1') AND (hall_b_reg = '1') AND (hall_c_reg = '0') THEN
        IF (direction = '0') THEN
                phase_a_percent_reg <= pwm_percent_reg;
                phase_b_percent_reg <= (others => '0');
                phase_c_percent_reg <= (others => '0');
                phase_a_reset_reg <= '0';
                phase_b_reset_reg <= '1';
                phase_c_reset_reg <= '0';
        ELSE
                phase_a_percent_reg <= (others => '0');
                phase_b_percent_reg <= (others => '0');
                phase_c_percent_reg <= pwm_percent_reg;
                phase_a_reset_reg <= '0';
                phase_b_reset_reg <= '1';
                phase_c_reset_reg <= '0';
        END IF;
END IF;
IF (hall_a_reg = '0') AND (hall_b_reg = '1') AND (hall_c_reg = '0') THEN
        IF (direction = '0') THEN
                phase_a_percent_reg <= (others => '0');
                phase_b_percent_reg <= pwm_percent_reg;
                phase_c_percent_reg <= (others => '0');
                phase_a_reset_reg <= '1';
                phase_b_reset_reg <= '0';
                phase_c_reset_reg <= '0';
        ELSE
                phase_a_percent_reg <= (others => '0');
                phase_b_percent_reg <= (others => '0');
                phase_c_percent_reg <= pwm_percent_reg;
                phase_a_reset_reg <= '1';
                phase_b_reset_reg <= '0';
                phase_c_reset_reg <= '0';
        END IF;
END IF;
IF (hall_a_reg = '0') AND (hall_b_reg = '1') AND (hall_c_reg = '1') THEN
        IF (direction = '0') THEN
                phase_a_percent_reg <= (others => '0');
                phase_b_percent_reg <= pwm_percent_reg;
                phase_c_percent_reg <= (others => '0');
                phase_a_reset_reg <= '0';
                phase_b_reset_reg <= '0';
                phase_c_reset_reg <= '1';
        ELSE
                phase_a_percent_reg <= pwm_percent_reg;
                phase_b_percent_reg <= (others => '0');
                phase_c_percent_reg <= (others => '0');
                phase_a_reset_reg <= '0';
                phase_b_reset_reg <= '0';
                phase_c_reset_reg <= '1';
        END IF;
END IF;
IF (hall_a_reg = '0') AND (hall_b_reg = '0') AND (hall_c_reg = '1') THEN
        IF (direction = '0') THEN
                phase_a_percent_reg <= (others => '0');
                phase_b_percent_reg <= (others => '0');
                phase_c_percent_reg <= pwm_percent_reg;
                phase_a_reset_reg <= '0';
                phase_b_reset_reg <= '1';
                phase_c_reset_reg <= '0';
        ELSE
                phase_a_percent_reg <= pwm_percent_reg;
                phase_b_percent_reg <= (others => '0');
                phase_c_percent_reg <= (others => '0');
                phase_a_reset_reg <= '0';
                phase_b_reset_reg <= '1';
                phase_c_reset_reg <= '0';
        END IF;
END IF;
```

```
                        IF (hall_a_reg = '1') AND (hall_b_reg = '0') AND (hall_c_reg = '1') THEN
                                IF (direction = '0') THEN
                                        phase_a_percent_reg <= (others => '0');
                                        phase_b_percent_reg <= (others => '0');
                                        phase_c_percent_reg <= pwm_percent_reg;
                                        phase_a_reset_reg <= '1';
                                        phase_b_reset_reg <= '0';
                                        phase_c_reset_reg <= '0';
                                ELSE
                                        phase_a_percent_reg <= (others => '0');
                                        phase_b_percent_reg <= pwm_percent_reg;
                                        phase_c_percent_reg <= (others => '0');
                                        phase_a_reset_reg <= '1';
                                        phase_b_reset_reg <= '0';
                                        phase_c_reset_reg <= '0';
                                END IF;
                        END IF;
                        IF (hall_a_reg = '0') AND (hall_b_reg = '0') AND (hall_c_reg = '0') THEN
                                phase_a_percent_reg <= (others => '0');
                                phase_b_percent_reg <= (others => '0');
                                phase_c_percent_reg <= (others => '0');
                                phase_a_reset_reg <= '1';
                                phase_b_reset_reg <= '1';
                                phase_c_reset_reg <= '1';
                        END IF;
                        IF (hall_a_reg = '1') AND (hall_b_reg = '1') AND (hall_c_reg = '1') THEN
                                phase_a_percent_reg <= (others => '0');
                                phase_b_percent_reg <= (others => '0');
                                phase_c_percent_reg <= (others => '0');
                                phase_a_reset_reg <= '1';
                                phase_b_reset_reg <= '1';
                                phase_c_reset_reg <= '1';
                        END IF;
                END IF;
        END IF;
    END PROCESS;
END ARCHITECTURE;
```

PWM_DEAD_TIME.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY PWM_DEAD_TIME IS
GENERIC
(
   CLOCK_DIVIDER    : INTEGER range 0 to 4095    := 2000;
   DEAD_TIME        : INTEGER range 0 to 4095    := 5
);
PORT
(
   clk                          : IN    STD_LOGIC;
   reset                : IN  STD_LOGIC;
   pwm_reset                    : IN  STD_LOGIC;
   pwm_percent          : IN  STD_LOGIC_VECTOR (6 downto 0);
   pwm_high_sw          : OUT STD_LOGIC;
   pwm_low_sw                   : OUT STD_LOGIC
);
END ENTITY;
ARCHITECTURE behavioral OF PWM_DEAD_TIME IS
   SIGNAL    pwm_percent_reg    : INTEGER range 0 to 127;
   SIGNAL    pwm_counter        : INTEGER range 0 to 4095;
BEGIN
   PROCESS (clk,pwm_reset)
   BEGIN
   IF (clk'EVENT AND clk = '1') THEN
            IF (reset = '0') THEN
                     pwm_high_sw <= '0';
                     pwm_low_sw <= '0';
                     pwm_percent_reg <= 0;
                     pwm_counter <= 0;
            ELSE
                     pwm_percent_reg <= to_integer(UNSIGNED(pwm_percent));
                     IF (pwm_reset = '1') THEN
                              pwm_high_sw <= '0';
                              pwm_low_sw <= '0';
                              pwm_counter <= 0;
                     ELSE
                              IF (pwm_counter < (CLOCK_DIVIDER-1)) THEN
                                       pwm_counter <= pwm_counter + 1;
                              ELSE
                                       pwm_counter <= 0;
                              END IF;
                              IF (pwm_percent_reg > 0) AND (pwm_percent_reg < 100) THEN
                                       IF  (pwm_counter  >  (((CLOCK_DIVIDER/100)  *  pwm_percent_reg)  +  DEAD_TIME))
THEN
                                                pwm_low_sw <= '1';
                                       ELSE
                                                pwm_low_sw <= '0';
                                       END IF;

                                       IF  (pwm_counter  >  DEAD_TIME)  AND  (pwm_counter  <  ((CLOCK_DIVIDER/100)  *
pwm_percent_reg)) THEN
                                                pwm_high_sw <= '1';
                                       ELSE
                                                pwm_high_sw <= '0';
                                       END IF;
                              ELSE
                                       IF (pwm_percent_reg = 0) THEN
                                                pwm_low_sw <= '1';
                                                pwm_high_sw <= '0';
                                       ELSE
                                                pwm_low_sw <= '0';
                                                pwm_high_sw <= '1';
                                       END IF;
                              END IF;
                     END IF;
            END IF;
   END IF;
   END PROCESS;
END ARCHITECTURE;
```

BAR_GRAPH.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
ENTITY BAR_GRAPH IS
PORT
(
   clk                              : IN    STD_LOGIC;
   reset                    : IN  STD_LOGIC;
   input                    : IN  STD_LOGIC_VECTOR (23 downto 0);
   led1                             : OUT  STD_LOGIC;
   led2                             : OUT  STD_LOGIC;
   led3                             : OUT  STD_LOGIC;
   led4                             : OUT  STD_LOGIC
);
END ENTITY;
ARCHITECTURE behavioral OF BAR_GRAPH IS
   SIGNAL input_reg         : INTEGER range 0 to 16777215;
BEGIN
   PROCESS (clk,reset)
   BEGIN
          IF (clk'EVENT AND clk = '1') THEN
                IF (reset = '0') THEN
                      led1 <= '0';
                      led2 <= '0';
                      led3 <= '0';
                      led4 <= '0';
                      input_reg <= 0;
                ELSE
                      input_reg <= to_integer(unsigned(input));
                      IF (input_reg > 20) THEN
                            led1 <= '1';
                      ELSE
                            led1 <= '0';
                      END IF;
                      IF (input_reg > 40) THEN
                            led2 <= '1';
                      ELSE
                            led2 <= '0';
                      END IF;
                      IF (input_reg > 60) THEN
                            led3 <= '1';
                      ELSE
                            led3 <= '0';
                      END IF;
                      IF (input_reg > 80) THEN
                            led4 <= '1';
                      ELSE
                            led4 <= '0';
                      END IF;
                END IF;
          END IF;
   END PROCESS;
END ARCHITECTURE;
```